AD A069604

DDC FILE COPY

ARI TECHNICAL REPORT
TR-78-A22

A Comparative Study of Flowcharts and Program Design Languages for the Detailed Procedural Specification of Computer Programs

LEVEL II

by

H. Rudy Ramsey and Michael E. Atwood
SCIENCE APPLICATIONS, INC.

and

James R. Van Doren
OKLAHOMA STATE UNIVERSITY

SCIENCE APPLICATIONS, INC.
7935 East Prentice Avenue
Englewood, Colorado 80110

THIS DOCUMENT IS BEST QUALITY PRACTICABLE.
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.

DDC
RECEIVED
JUN 11 1979
A

SEPTEMBER 1978

Contract DAHC 19-76-C-0040

Monitored technically by Edgar M. Johnson and Jean Nichols Hooper
Battlefield Information Systems Technical Area, ARI

Prepared for

U.S. ARMY RESEARCH INSTITUTE
for the BEHAVIORAL and SOCIAL SCIENCES
5001 Eisenhower Avenue
Alexandria, Virginia 22333

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Approved for public release; distribution unlimited

79 06 08 027

U. S. ARMY RESEARCH INSTITUTE
FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency under the Jurisdiction of the Deputy Chief of Staff for Personnel

JOSEPH ZEIDNER
Technical Director

WILLIAM L. HAUSER
Colonel, US Army
Commander

Research accomplished for the
Department of the Army

Science Applications, Inc.

NOTICES

DISTRIBUTION: Primary distribution of this report has been made by ARI. Please address correspondence concerning distribution of reports to: U. S. Army Research Institute for the Behavioral and Social Sciences, ATTN: PERI-P, 5001 Eisenhower Avenue, Alexandria, Virginia 22333.

FINAL DISPOSITION: This report may be destroyed when it is no longer needed. Please do not return it to the U. S. Army Research Institute for the Behavioral and Social Sciences.

NOTE: The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR-78-A22 (19) (18) ARI	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: (9)	
4. TITLE (and Subtitle): A COMPARATIVE STUDY OF FLOWCHARTS AND PROGRAM DESIGN LANGUAGES FOR THE DETAILED PROCEDURAL SPECIFICATION OF COMPUTER PROGRAMS (6)	5. TYPE OF REPORT & PERIOD COVERED: Technical Report, 12 Jul 1976-11 Nov [redacted]1977
	6. PERFORMING ORG. REPORT NUMBER: (14) SAI-78-078-DEN
7. AUTHOR(s): (10) H. Rudy Ramsey, Michael E. Atwood & James R. Van Doren	8. CONTRACT OR GRANT NUMBER(s): (15) DAHC19-76-C-0040
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Science Applications, Inc. 7935 E. Prentice Avenue Englewood, Colorado 80110 (16)	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 2Q762725A778 (12) 173 p.
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Institute for the Behavioral and Social Sciences (PERI-OS) 5001 Eisenhower Avenue, Arlington, Virginia 22333 (11)	12. REPORT DATE: Sept [redacted]1978
	13. NUMBER OF PAGES: 103
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Monitored technically by Edgar M. Johnson and Jean Nichols Hooper, Battlefield Information Systems Technical Area, ARI.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computer programming, flowcharting, computer program documentation, specifications.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An experiment was performed to assess the relative merits of Program Design Languages (PDLs) and flowcharts as techniques for the development and documentation of detailed designs for computer programs.

Twenty students in a computer science graduate course participated in this experiment. Working individually, the students designed a two-pass assembler for a simple minicomputer. Half the students expressed their design → next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

392 878

for the first pass of the assembler in the form of a flowchart, and expressed their design for the second pass in a Program Design Language. The other half of the students used a PDL for pass one, and a flowchart for pass two. Flowcharts and PDLs were compared on the basis of various measures of overall design quality, design errors, level of detail of designs, time expended in developing designs, and subjective preferences.

Having completed this design task, the subjects then performed an implementation task. They were given fairly detailed procedural designs for a program which simulates the function of a fairly sophisticated minicomputer. They were then required to develop a working version of the program in PL/1. Although the designs were logically equivalent, half the students received their simulator design in flowchart form, and half in PDL form. Flowcharts and PDLs were compared on the basis of design comprehension test performance, various measures of overall implementation quality, implementation errors, and subjective preferences.

In the context in which this study was performed, the use of a Program Design Language (PDL) by a software designer, for the development and description of a detailed program design, produced better results than did the use of flowcharts. Specifically, the designs appeared to be of significantly better quality, involving more algorithmic or procedural detail, than those produced using flowcharts. In addition, flowchart designs exhibited considerably more abbreviation and other space-saving practices than did PDL designs, with a possible adverse effect on their readability.

When equivalent, highly readable designs were presented to subjects in both PDL and flowchart form, no pattern of short-term or long-term differences in comprehension of the design was observed. No significant differences were detected in the quality or other properties of programs written as implementations of the designs. Subjective ratings indicated a mild preference for PDLs.

Overall, the results suggest that software design performance and designer-programmer communication might be significantly improved by the adoption of informal Program Design Languages, rather than flowcharts, as a standard documentation method for detailed computer program designs.

FOREWORD

The Battlefield Information Systems Technical Area is concerned with the human resource demands of increasingly complex battlefield systems used to acquire, transmit, process, disseminate, and utilize information. This increased complexity places greater demands upon the operator interacting with the machine system. Research in this area is focused on human performance problems related to interactions within command and control centers as well as issues of system development.

It is concerned with such areas as software development, topographic products and procedures, tactical symbology, user oriented systems, information management, staff operations and procedures, and sensor systems integration and utilization.

One area of special interest involves the development of computer software to support automated battlefield systems. Software development is a costly, unreliable, not well understood process. The present research assessed the relative merits of Program Design Languages (PDLs) and flowcharts as techniques for the development and documentation of detailed designs for computer programs. The research is part of a larger effort to develop a conceptualization of the programming process and identify behavioral bottlenecks in software development. Efforts in this area are directed at improving accuracy and productivity in programming through the design of procedures, languages, and methods to enhance programmer performance.

Research in the area of human factors in software development is conducted as an in-house effort augmented contractually by organizations selected as having unique capabilities and facilities. The effort is responsive to requirements of Army Project 2Q762725A778, and to general requirements expressed by members of the Integrated Software Research and Development Working Group (ISRAD).

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or special
A 23

BRIEF

Requirement:

This experimental study was performed in an attempt to assess the relative merits of Program Design Languages (PDLs) and flowcharts as techniques for the development and documentation of detailed designs for computer programs.

Procedure:

Twenty students in a computer science graduate course participated in an experiment. Working individually, the students designed a two-pass assembler for a simple minicomputer. Half the students expressed their design for the first pass of the assembler in the form of a flowchart, and expressed their design for the second pass in a Program Design Language. The other half of the students used a PDL for pass one, and a flowchart for pass two. Flowcharts and PDLs were compared on the basis of various measures of overall design quality, design errors, level of detail of designs, time expended in developing designs, and subjective preferences.

Having completed this design task, the subjects then performed an implementation task. They were given fairly detailed procedural designs for a program which simulates the function of a fairly sophisticated minicomputer. They were then required to develop a working version of the program in PL/1. Although the designs were logically equivalent, half the students received their simulator design in flowchart form, and half in PDL form. Flowcharts and PDLs were compared on the basis of design comprehension test performance, various measures of overall implementation quality, implementation errors, and subjective preferences.

Findings:

In the context in which this study was performed, the use of a Program Design Language (PDL) by a software designer, for the development and description of a detailed program design, produced better results than did the use of flowcharts. Specifically, the designs appeared to be of significantly better quality, involving more algorithmic or procedural detail, than those produced using flowcharts. In addition, flowchart designs exhibited considerably more abbreviation and other space-saving practices than did PDL designs, with a possible adverse effect on their readability.

When equivalent, highly readable designs were presented to subjects in both PDL and flowchart form, no pattern of short-term or long-term differences in comprehension of the design was observed. No significant differences were detected in the quality or other properties of programs written as implementations of the designs. Subjective ratings indicated a mild preference for PDLs.

Utilization of Findings:

Overall, the results suggest that software design performance and designer-programmer communication might be significantly improved by the adoption of informal Program Design Languages, rather than flowcharts, as a standard documentation method for detailed computer program designs.

TABLE OF CONTENTS

	Page
INTRODUCTION	1
METHOD	10
Overview	10
Subjects	11
Preliminary Phase	12
Design Phase	14
Implementation Phase	16
RESULTS	21
Design Phase	21
Time to Produce Designs	22
Quality and Properties of Designs	24
Subjective Ratings of Documentation Methods	35
Implementation Phase	39
Comprehension of Simulator Design	39
Quality of Implementation	42
Subjective Ratings of Documentation Methods	45
DISCUSSION	49
CONCLUSIONS	54
REFERENCES	55
APPENDIX - A Schedule of Events	A-1
APPENDIX - B Consent Forms	B-1
APPENDIX - C Background Questionnaire	C-1
APPENDIX - D Software Design Methods Handout	D-1
APPENDIX - E PDL and Flowchart Examples	E-1
APPENDIX - F Basic Description of Design Problem	F-1
APPENDIX - G Description of Pass 1 Design Problem	G-1
APPENDIX - H Description of Pass 2 Design Problem	H-1
APPENDIX - I Time Expenditure Questionnaire, Design Phase	I-1
APPENDIX - J PDL-Flowchart Questionnaire	J-1
APPENDIX - K Simulator Design, Flowchart Form	K-1
APPENDIX - L Simulator Design, PDL Form	L-1
APPENDIX - M Simulator Design Revision Memo	M-1
APPENDIX - N Design Comprehension Test	N-1
APPENDIX - O Time Expenditure Questionnaire, Implementation Phase	O-1
APPENDIX - P Examples of Subject-Generated Designs	P-1
APPENDIX - Q Example of Job-Generated Simulator Program	Q-1

TABLES

		Page
Table 1.	Summary of Experimental Conditions	18
Table 2.	Means and (in Parentheses) Standard Deviations of Reported Design Times (hours)	23
Table 3.	Means and (Standard Deviations) of Design Quality Ratings	26
Table 4.	Analysis of Variance of Total Design Scores	29
Table 5.	Means and (in Parentheses) Standard Deviations of "Design Style" and Level-of-Detail Measures	32
Table 6.	Summary of Abbreviations Used in Pass 1 Designs	34
Table 7.	Means and (in Parentheses) Standard Deviations of Questionnaire Ratings after Design Phase	38
Table 8.	Results of Multiple-Choice Portion of Design Comprehension Test	40
Table 9.	Means and (Standard Deviations) of Implementation Quality Ratings	44
Table 10.	Means and (Standard Deviations) of Questionnaire Ratings after Implementation Phase	47

FIGURES

		Page
Figure 1.	An Example of a PDL Specification	6
Figure 2.	Grading Criteria for Assembler Designs	25
Figure 3.	Mean Total Design Score, by Group	27
Figure 4.	Grading Criteria for Implemented Simulators	42

PRECEDING PAGE BLANK-NOT FILMED

INTRODUCTION

In recent years, several trends have emerged which have caused the Department of Defense to direct increasing attention toward the investigation of the behavioral factors which affect the development of computer software. DoD software expenditures, which probably now exceed $5 billion annually, have been increasing both in absolute terms and as a percentage of total system development costs (Boehm, 1973). At the same time, the increasing size and complexity of software systems has caused software reliability to become a dominating problem in many modern weapon and support systems (Boehm & Haile, 1972). During the 1970s, there has been an emerging awareness, both in DoD and in the computer science community at large, that the control of software costs and quality will depend ultimately on an understanding of the complex behavioral factors affecting software design, programming, documentation, etc.

The quality of software documentation can significantly influence both the quality and cost of the software (to the extent that the documentation is used during the development effort) and its maintainability and modifiability. Software documentation is an area which has long been subject to DoD standards. A number of new software documentation methods have recently emerged (e.g., HIPO charts, structure charts, program design languages). DoD is now considering recommendations that some of these techniques be incorporated more widely in its standards. One of these techniques (Program Design Languages) is the subject of this study.

The importance of documentation seems to be widely recognized in the software development community. For example, when programming managers were asked to identify, via a two-stage Delphi technique, those variables which most affect programmer productivity, documentation quality was rated the most important factor (Scott & Simmons, 1974). Yet, in spite of this

consensus about the importance of good documentation, there is considerable disagreement concerning the form it should take.

One of the reasons for the wide diversity of documentation types is the diversity of purposes for which software documentation can be used. Judd (1972) advocates the development of program design documentation, error diagnosis documentation, operational documentation, maintenance documentation, development documentation, and marketing documentation, all as separate products of a software development effort. It is undoubtedly true that documentation methods appropriate for some of these purposes may not be appropriate for others. This report focuses upon "detailed program design" documentation, typically developed by a software system designer and given to a programmer as the specification of his programming task. It is the formal medium of communication between the designer and the programmer.

Flowcharts have long been accepted in the computer science community as the standard medium for detailed program design documentation, as well as for archival documentation for later use by maintenance programmers. Recently, however, the use of flowcharts for archival documentation has been questioned by a number of critics, who suggest that flowcharts may not aid program comprehension (Weinberg, 1971), or error diagnosis (Aron, 1974), and that they are an unnecessary drain on project resources (Brooks, 1975). White (1975) has suggested that modern programming languages provide a more powerful and concise documentation medium, in their own right, than do flowcharts which attempt to convey the procedural details of a program.

Shneiderman et al. (1977) performed a series of experiments to determine whether detailed flowcharts might improve the programmer's performance in any of several programming tasks. Flowcharting before

coding did not improve the quality of the resulting program, as graded by a graduate assistant. When programs were presented with or without flowcharts, flowcharts did not result in significantly improved performance on either comprehension tests or modification tasks. In one study, comprehension even appeared to be degraded by the presence of a flowchart. In a study in which some subjects had prior experience with flowcharts while others did not, the experienced subjects performed somewhat better with flowcharts, while the inexperienced subjects did not.

Flowcharts and similar documentation methods have also been studied outside the software context. Kammann (1975) reported an experiment involving a telephone dialing task in which subjects (housewives and technical personnel) performed better when it was presented in the form of a flowchart than when presented in the form of a text description. An experiment by Wright and Reid (1973) has identified what may be a key task variable in these studies. In their experiment, subjects were required to memorize a procedure or to execute the procedure by hand without necessarily memorizing it. When the subjects hand-executed the procedure while its description was still available to them, performance was better for procedures described by flowchart or decision table than for those described by prose or a series of "short sentences." But when the subject's task was execution of the procedure from memory, "short sentences" were best, and performance with flowcharts was rather poor. The authors suggest that rehearsal effects may account for this difference, since material presented verbally may be more readily rehearsed.

Wright and Reid suggest that information presented in flowchart form is, at least in part, encoded visually, with the result that successful transfer to long-term memory is hindered. On the other hand, information presented in short-sentence form is encoded verbally and is more likely to be successfully integrated into long-term memory. The mechanism suggested by Wright and Reid (dual encoding with differential rehearsal effects)

seems plausible (cf. Craik, 1973; Craik and Lockhart, 1972; Paivio, 1969). In any event, the study certainly suggests that these two information presentation forms have different retention patterns. It should be noted, however, that exposure time was limited in this study, and that the two presentation forms, though equal in logical information content, differed somewhat in information structure.

In general these results suggest that flowcharts may aid hand-execution of a procedure, but may not aid comprehension of the procedure. Shneiderman et al. (1977) present some data which support this hypothesis. In one study, they report separately "comprehension" questions which involve hand execution and questions which involve interpretation of the programs. Subjects who were given both flowcharts and programs did about as well on hand-execution items as those who were given only the programs. On interpretation items, though, the program-only group performed somewhat better.

Unfortunately, it may not be legitimate to generalize these findings to large software projects or to summary design documentation. Those studies which have addressed programming tasks have not only been restricted to relatively inexperienced student programmers, but have also investigated only relatively small, simple programs. Even if the program itself constitutes a better form of archival documentation than does a flowchart, the size of a program may be a significant factor. Beyond a certain size, a program(s) cannot be read directly by one person, and summary documentation of some sort is, therefore, required in large software projects. Whether flowcharts or some other form are best for this purpose is not known. Even for programs of intermediate size, the reader might be aided by such summary documentation.

Although the referenced studies have been concerned primarily with archival documentation, rather than with designer-to-programmer

design communication, they may have implications for the latter. The data suggest that a computer program, expressed in a high-level programming language, is more comprehensible than the corresponding flowchart. If this is true, then an artificial design language, with a programming-language-like syntax, might also be preferable to flowcharts for the expression of software design information. Such languages, commonly called Program Design Languages (PDLs) have been discussed in the literature for several years. PDLs have gradually evolved from the rather informal use of "pseudo-code" to express designs or algorithms. Gradually, this form of documentation became more formal, evolving into a PDL form. Although a few presumably successful case studies have been reported without substantiating data (e.g., Van Leer, 1976), the authors are unaware of any experimental studies bearing directly on the issue of PDL efficacy.

Figure 1 is an example of PDL specification for a program which computes social security withholding (FICA) amounts from a payroll data base, and prints a report of those values. This example is taken from a report (Kraly et al., 1975) concerning program design methods and documentation techniques recommended for adoption by DoD.

PDLs can vary greatly in the degree of formality. At one extreme, is a "freeform" PDL, such as that in Figure 1. In a freeform PDL, there is a minimum of syntactic and semantic constraints; the terms and expressions that are used are determined by the user. In a "formal" PDL, the syntax and semantics are highly constrained; the user is restricted to using keywords (e.g. READ, WRITE, etc.) and elements of an expression may be syntactically identified by underlines, parentheses, etc. In general, a formal PDL is more precise than an informal PDL, but it is also more difficult to learn and use. Formal PDLs tend to be preferred especially when the design specification is subject to direct computer processing, as for consistency checking.

```
PRINT FICA REPORT HEADER
OBTAIN FICA PERCENT AND FICA LIMIT FROM CONTRAINTS FILE
SET FICA TOTAL TO ZERO
DO FOR EACH RECORD IN SALARY FILE
    OBTAIN EMPLOYEE NUMBER AND TOTAL SALARY TO DATE
    IF TOTAL SALARY IS LESS THAN FICA LIMIT THEN
        SET FICA VALUE TO TOTAL SALARY TIMES FICA PERCENT
    ELSE
        SET FICA VALUE TO FICA LIMIT TIMES FICA PERCENT
    ENDIF
    PRINT EMPLOYEE NUMBER AND FICA VALUE
    ADD FICA VALUE TO FICA TOTAL
ENDDO
PRINT FICA TOTAL
```

Figure 1. An Example of a PDL Specification

There are several reasons for believing that PDLs and flowcharts may differentially affect performance in design, design documentation, and design comprehension tasks. First, information presented in these two media may be encoded in memory in different ways, at least with limited exposure time. Notice that PDL design specifications are very similar to the "short sentences" used by Wright and Reid (1973), while flowcharts are similar to their "algorithms."

Secondly, PDL and flowchart forms may differ in the processing effort required to encode them in memory even if they are encoded similarly. PDL forms are not only already verbal (and thus, perhaps, more easily integrable into long-term memory), but also appear to be more compatible with the hierarchic propositional memory representation hypothesized by Atwood and Ramsey (in press) for program comprehension. Program documentation which has a similar structure to the representation of the program in memory may require less processing in order to be encoded in memory. Given sufficient processing time, however, flowchart information might be encoded in the same way as PDL information, even if they are encoded differently with brief exposure times, as suggested by Wright and Reid in (1973) results. In most actual software development situations, exposure time is not highly constrained; thus, if this is the only difference between PDLs and flowcharts from the point of view of comprehension, it may result in only a mild preference for one technique over the other.

Third, PDLs and flowcharts may emphasize different properties of the underlying software design. At an obvious level, flowcharts appear to emphasize flow of control, while PDLs may have a greater emphasis on program structure. This may affect the performance of the designer, as well as the programmer. From the designer's viewpoint, it may be that different classes of design problems are more easily detectable with different design documentation formats. More basically, it

may simply be easier to express the properties which the designer considers important in one way than in the other. It is even conceivable that some types of design are more amenable to PDLs, while others are more compatible with flowcharts.

From the viewpoint of the programmer, the form in which the programming problem is expressed may very well affect his basic understanding of the problem. In various problem-solving tasks, it has been shown that the perceptual process is important to the formation of initial conceptual problem structures (Simon & Barenfield, 1969), and that the form in which a problem is expressed can thereby affect problem-solving strategy (Simon & Hayes, 1976) and performance (Jeffries et al, 1977; Simon & Hayes, 1976).

Fourth, and finally, it seems fairly clear that PDLs have some practical advantages, such as decreased production costs and easier maintainability, relative to flowcharts and other graphical techniques (Ortega, 1974).

Thus, an analytical comparison of PDLs and flowcharts would appear, overall, to favor PDLs for detailed design documentation. Only empirical evaluation, however, can provide really convincing evidence in favor of one or another technique. The following sections will present an experimental study performed to obtain such evidence.

The experiment reported here was intended to provide empirical evidence which might assist in the selection of flowcharts or PDLs as a medium for detailed program design documentation. The experiment was intended to investigate the effects of documentation format both on performance of the designer in a software design task and on performance of the programmer in a program implementation task. Although the study was confined to one-person design and implementation efforts, the tasks

(especially the implementation task) were much larger than those which have typically been employed in behavioral studies of software development, and were fairly "realistic" in terms of both program size and programmer experience. The study attempted to observe the effects of documentation format on design quality, detailed design properties, designer effort, design comprehension by the programmer, resulting program quality and error properties, programmer effort, and subjective preferences of the participants.

METHOD

OVERVIEW

Twenty students in a computer science graduate course participated in the experiment. Working individually, the students designed a two-pass assembler for a simple minicomputer. Half the students expressed their design for the first pass of the assembler in the form of a flowchart, and expressed their design for the second pass in a PDL. The other half of the students used a PDL for pass one, and a flowchart for pass two. This 2 X 2 repeated-measures Latin square design was employed in an effort to correct for individual differences and to increase the statistical power which could be achieved with the limited number of subjects available. Although this design has the disadvantage that any chance group difference is confounded with interactions, it was felt that the advantages outweighed this disadvantage. Flowcharts and PDLs were compared on the basis of various measures of overall design quality, design errors, level of detail of designs, time expended in developing designs, and subjective preferences.

Having completed this design task, the subjects then performed an implementation task. They were given fairly detailed procedural designs for a program which simulates the function of a fairly sophisticated minicomputer. They were then required to develop a working version of the program in PL/1. Although the designs were logically equivalent, half the students received their simulator design in flowchart form, and half in PDL form. Subjects from each design-phase group were divided equally into these two implementation-phase groups. Flowcharts and PDLs were compared on the basis of design comprehension test performance, various measures of overall implementation quality, implementation errors, time expended in developing implementation, and subjective preferences.

SUBJECTS

Subjects were 1 female and 19 male graduate students at Oklahoma State University (OSU). With one exception, subjects were Master of Science candidates majoring in Computer Science and enrolled in the course, "Computer Structure and Programming," described below. One student, who later withdrew from the course, was majoring in engineering. Because of the location of OSU away from large urban centers, and possibly other factors, students in this curriculum tend to be full-time students with no outside (non-university) employment. Subjects ranged in age from 21 to 40, with a mean age of 26.1 years. They had completed, on the average, 8.9 computer science courses (undergraduate and/or graduate) in addition to current enrollment (range 4-16).

Based on previous knowledge of the faculty, two subject factors were identified as sufficiently relevant to the subjects' performance in the study to warrant a special effort to balance the effects of these factors across experimental groups. These factors were native language (a significant number of foreign students attend OSU) and participation in a particular graduate student curriculum, discussed below. Five of the subjects were not native speakers of English. Two of these five were considered completely fluent in English by their instructor. The remaining three were considered competent in English; all three had taken prior degrees at English-language universities. As a precautionary measure, however, these three subjects were assigned (at random) to different experimental groups.

Three of the subjects were pursuing a specialized curriculum (the "Programming Language Option"), which emphasizes compilers, assemblers, and machine simulators. Since the design and programming problems used in the experiment involved an assembler and a machine simulator, these three subjects were assigned (at random) to different

experimental groups. With the two exceptions just described, subject assignment to experimental groups was entirely random.

The particular course in which the experiment was performed was entitled, "Computer Structure and Programming." As described in the department's brochure, the course covers the areas of:

> Computer structure, flow of control, arithmetic and logical operations. Machine instructions, indexing, addressing, linkages and input-output. Assembly, macro and emulation-simulation systems.

This particular required course has a reputation, among the students, of being the most difficult course in the Master's curriculum.

PRELIMINARY PHASE

The first phase of the experiment was introductory in nature and involved the same procedures for all students. The experiment began at a class session approximately 3 weeks after the start of the course (a detailed schedule of events can be found in Appendix A). In this class session, the students were given an introduction to the experiment, as well as an introduction to the concept of Program Design Languages, and were asked to provide background information describing their computer science experience and some personal characteristics.

The lecture began with a brief discussion of the nature of the experiment. A concerted effort was made to introduce the experiment in an unbiased way. Statements of specific hypotheses were avoided, and the experiment was described as an attempt to compare two techniques, each of which probably has both advantages and disadvantages. It was pointed out that participation was entirely voluntary, that the experiment was entirely consistent with course goals and would involve little additional effort by the subjects, and that subjects completing all

phases of the experiment would be paid $25. A more detailed impression of the way in which the experiment was described to the students can be gained by reading the consent form which they were given (see Appendix B).

Because flowcharts were the standard mechanism for the procedural description of programs in OSU courses prior to this experiment, most of the subjects had had little prior exposure to PDLs. Thus, it was necessary to provide them with sufficient training in the use of PDLs that they would be able to generate and comprehend specifications expressed in this form. At the same time, it was obviously desirable to minimize the biasing effect that might result from a training session devoted purely to a discussion of PDLs. We attempted to satisfy both of these objectives by presenting parallel discussions of PDLs and flowcharts. After the basic concept of a Program Design Language had been introduced, further discussion was directed primarily toward illustrating how a particular construct might be expressed, both in PDL and in flowchart form. This presentation took the form of a discussion of two handouts (Appendices D and E) which were given to the students at the start of the class session. The discussion was recorded on audio tape for later reference.

Appendix D served the dual functions of introducing the concept of a PDL to the student and of defining the particular PDL (and flowchart) conventions which were to be used in both the design and implementation stages of the study. The particular PDL used was a relatively unstructured or "freeform" PDL (cf. Kraly et al, 1975). That is, the user was required to use a particular basic syntax involving a particular set of basic structural constructs (PROCEDURE...END, IF...THEN...ELSE, DO...END, etc.), but was relatively unconstrained with respect to the particular statements which might appear within these basic structures. He might say INCREMENT ITERATION_COUNT, ADD ONE TO ITERATION_COUNT, UPDATE ITERATION_COUNT, ITERATION_COUNT = ITERATION_COUNT +1, etc. The

particular PDL used in the study was chosen to correspond in many respects to features of PL/1, with which students were familiar. Similarly, the basic set of flowchart symbols available to the user was specified, and consisted of the flowchart symbols in general use in OSU computer science graduate courses. The particular statements which might appear within flowchart symbols were relatively unconstrained, however.

The second handout (Appendix E) contained the specification for a very simple minicomputer simulator, expressed in both flowchart and PDL form. This handout, and its discussion, served not only to provide a concrete example of both flowchart and PDL specifications, but also to introduce the students to some of the concepts involved in machine simulators.

When the PDL-flowchart presentation had been completed, each student was given two consent forms (Appendix B) and a background questionnaire (Appendix C). The students were asked to sign the consent forms, fill out the questionnaire, and return them at the following class session, if they wished to participate in the experiment. All 20 students agreed to participate, and returned all of the requested forms.

DESIGN PHASE

In the design phase of the experiment, the subjects were given the task of designing a two-pass assembler for a minicomputer. The particular minicomputer involved was a hypothetical machine (the "small computer" which had been defined and used as an example earlier in the course. The subjects were presumably thoroughly familiar with the hardware instruction set of this machine. Earlier in the course they had been instructed in the use of the programming language APL as a notation for concise, rigorous definition of hardware instruction sets. As an exercise, they had then developed a complete specification, in APL, of the instruction set of this particular machine.

In addition to this familiarization with the machine, the students had been given instructions concerning the design of assemblers as part of the course in which the experiment was conducted. No prior discussion had occurred with respect to the possible properties of an assembler, or of an assembler language, for the "small computer," however. At the start of the design phase of the experiment, the subjects were given a handout (Appendix F) which described this assembler language, and which instructed the students that they would be required to design a two-pass assembler, documenting one pass via flowchart and the other pass via PDL.

Along with the general assembler design problem description, the subjects received a brief description of the responsibilities of the first pass of the assembler (Appendix G). These two handouts constituted the entire problem description for pass 1. All subjects designed the pass 1 program first, but half of the subjects were told to prepare their designs in flowchart form, and half in PDL form. A third handout (Appendix I) accompanied the problem description. This was a form on which the subjects were asked to record the amount of time expended on the project in each of several categories (initial design preparation, design revision, hand-checking, preparation of documentation, etc.). The subjects were told to prepare their designs and turn them in with the time questionnaire, whenever they were completed, at which time they would be given pass 2 instructions. They were told that they had at most five days to complete the first pass, and that the second pass design would be due seven days after the start of the design phase.

When a subject turned in his pass 1 design, he was given a description of the responsibilities of the pass 2 program (Appendix H), along with a time questionnaire essentially identical to that shown in Appendix I. Those subjects who had prepared their pass 1 designs in flowchart form were instructed to use the PDL form for pass 2, and vice-versa.

After completion of the pass 2 designs, subjects were given a take-home questionnaire (Appendix J) which asked them to subjectively compare flowcharts and PDLs with respect to their overall utility to the software designed and to the programmer, and a number of more detailed criteria.

In addition to the time and subjective preference data, finished designs were graded on overall quality by an expert judge, and were analyzed for the nature of design errors, level of detail of the design, and other properties. This judge was an associate professor on the computer science department faculty, very knowledgable about software design techniques and methods, and highly experienced in developing assemblers and translators.

IMPLEMENTATION PHASE

In the implementation phase, the subjects were given a program design, expressed in either flowchart or PDL form, and were required to develop a working version of the program in PL/1. This implementation effort was the major term project for this course. The program was a simulator for the Data General MicroNova minicomputer. As minicomputers go, the instruction set and execution behavior of the MicroNova are fairly complicated. The simulator, which was to run on an IBM 360, was intended to correctly execute MicroNova machine language programs, with the exception that console operations were excluded and input/output device control was somewhat simplified. In addition, the simulator design included some specialized capabilities (Trace, Dump, etc.) which were associated with the simulator itself.

Prior to the start of the experimental portion of the implementation phase, the students studied the properties of the MicroNova, in order to gain a thorough understanding of the machine itself. This study included lectures, reading of MicroNova reference documents (e.g., Data

General Corp., 1976), and, in some cases, additional literature research initiated by the students. Thus, at the time the simulator designs were distributed, the students were presumably thoroughly familiar with the MicroNova, but had no knowledge of the detailed simulator design, which described the program they were to implement.

Half the subjects were given a simulator design expressed in flowchart form (Appendix K), and half were given a design in PDL form (Appendix L). Although several minor design errors were uncovered later and were corrected by issuing an errata sheet (Appendix M), those errors were essentially equivalent in the two documentation forms. The errata sheet was issued 12 days after distribution of the designs. A strong effort was made to ensure complete logical equivalence of the two versions of the simulator design, and only one minor logical difference between the two versions has subsequently been detected. This difference was discovered and corrected 42 days after distribution of the designs.

In order to control for any residual differential effect of the earlier ("design phase") study which might bias the subjects' performance in this second experiment, the subjects from each of the earlier groups were divided equally between the two experimental groups used in this phase of the study. Considered overall, then, there were four separate groups of 5 subjects each (ignoring attrition), which were exposed to the experimental conditions illustrated in Table 1. Although all 20 subjects successfully completed the design stage of the experiment, 4 subjects withdrew from the course, or withdrew altogether from school, and therefore failed to complete the implementation phase, as indicated in the table.

Simulator designs were handed out at the beginning of a class session. The subjects were instructed to study the designs for 45 minutes in preparation for a comprehension test. Then, in an attempt to determine

TABLE 1. Summary of Experimental Conditions

Group	Number Starting	Number Completing	Order of Conditions in Design Phase	Condition in Implementation Phase
1	5	4	PDL-Flowchart	PDL
2	5	5	PDL-Flowchart	Flowchart
3	5	4	Flowchart-PDL	PDL
4	5	3	Flowchart-PDL	Flowchart

whether PDLs and flowcharts have any detectable differential effect on early design comprehension, a comprehension test (Appendix N) was administered. The subjects were allowed 35 minutes to complete this test. The test contained a variety of items intended to provide measures of several comprehension factors:

- understanding of MicroNova function
- ability to distinguish between MicroNova and simulator functions
- comprehension of modular structure of simulator design
- comprehension of procedural operation of simulator
- ability to pinpoint probable modular location of error causing specified simulator malfunction
- ability to specify changes needed to cause a specified change in simulator function.

(A second administration of this test, which was to have occurred after one week of study of the designs by the subjects, was inadvertently delayed until after the subjects had completed their program implementation task.)

After the designs were distributed, the subjects were given 8 1/2 weeks to develop a complete, working simulator. They were allowed to establish their own schedules for completing this task. Each subject was required to maintain, and turn in at the end of the project, a project folder. The folder contained the entire printout of each computer run made in connection with the project. Whenever new source programs were compiled, listings of those programs appear in these printouts, along with any error messages issued by the compiler, link/editor, etc. Input and output of simulator test runs were also included, as was the computer's job summary for each run. These printouts thus provide data for determination of simple summary statistics, such as number of runs, CPU utilization, etc., as well as more substantive analyses of error types and locations, etc.

Each project folder also contained a "run log," in which the students recorded the purpose of each run, nature of errors encountered, etc. This is a fairly standard practice at OSU, and was included in this project as an aid to later detailed analysis of the individual runs. Finally, the project folders contained a weekly time questionnaire (Appendix O) on which subjects indicated, by category, the time they had expended on the project in the preceding week.

In addition to the project folder, subjects turned in, at the end of the project, their finished simulator source programs in computer-readable form. A very detailed test procedure was run on each of these programs in an effort to assess correctness of simulator function and to detect those kinds of conceptual or semantic errors which the experimenters were able to anticipate a priori.

After the subjects had completed the implementation project and turned in their final programs and project folders, the simulator design comprehension test (Appendix N) and the PDL-Flowchart preference questionnaire (Appendix J) was readministered.

In addition to the various qualitative and quantitative measures already mentioned, the final simulator programs were rated on overall quality by an expert judge, (previously described) who also performed fairly detailed error analyses of the final programs.

RESULTS

DESIGN PHASE

The allocation to experimental groups of non-native-English-speakers and of students in a specialized programming-language curriculum was explicity constrained. These special inter-subject differences were dichotomous and affected small numbers of students, so that the assignment of two such subjects to a single group might significantly bias the study. Other differences were assumed to be controlled by the otherwise random subject-to-group assignment. Unfortunately, this random assignment resulted in a difference in average ability between the two groups in the design phase of the study. This difference is reflected, for example, in their Grade Point Averages (GPAs) for all computer science graduate courses.

There is at least a suggestion, in the available data, that this group difference may be related more to ability than to experience. The two groups are almost identical with respect to average number of computer science courses taken (8.7 for the group designing the first pass using a PDL versus 9.1 for the other group), average number of graduate computer science courses taken (4.2 versus 3.5), and number reporting past experience with translator implementation and related tasks (5 each).

For convenience of reference, let us refer to the group which used a PDL for pass 1 of the assembler, and flowcharts for pass 2, as the PDL-FL group. The other group will be referred to as the FL-PDL group. Then the mean GPA for the PDL-FL group was 2.986, while the FL-PDL mean was 3.587. Standard deviations were .606 and .334, respectively. Statistical tests indicate significant differences in both the means ($t(15) = 2.71$, $p < .02$) and variances ($F(9,8) = 3.30$, $p < .10$). The PDL-FL group included several students whose overall performance was relatively low, both within

this course and in graduate school in general, and who had no counterparts in the FL-PDL group, thus accounting for both differences.

This group difference does not greatly interfere with our ability to analyze the study results and draw reasonable conclusions. No such difference occurred between experimental groups in the implementation phase of the study. The use of a Latin square design in the design phase of the study sharply reduces the impact of the group difference on the analysis of that phase. In fact, the Latin-square analysis of variance employed for most of the design-phase analyses is conservative in this situation (McNemar, 1969, p. 387). The group difference should be kept in mind by the reader, however. If either flowchart or PDL is consistently preferable in the design of both pass 1 and pass 2 of the assembler, such a difference can be detected by statistical analysis. However, if for some reason a PDL is better for one pass and flowcharting for the other, perhaps due to differences in the design tasks, that interaction is confounded with the group differences in this study and cannot be detected. Based on a careful consideration of the design tasks in passes 1 and 2, no such interaction was expected.

Time to Produce Designs

Table 2 presents a summary of the students' responses to the design phase time expenditure questionnaire. On the average, the students reported spending about 7.6 hours on the pass 1 design task and about 5.4 hours on pass 2. The reported times for pass 1 include some general problem familiarization not repeated in pass 2. No statistical differences exist between PDL or flowchart conditions or between experimental groups on either the total time or any of its components. It should be noted that subjective reports of time expenditures are not generally a very reliable or sensitive measure. In any event, these data provide no basis for concluding that either documentation technique requires more time than the other.

Table 2. Means and (in Parentheses) Standard Deviations of Reported Design Times (hours)

	Group	Preparing Initial Design	Revising Design	Hand Checking Design	Preparing Documentation	Just Thinking About Project	Other	Total
Pass 1	PDL*	2.18 (1.15)	1.96 (1.46)	.66 (0.57)	.54 (0.63)	1.82 (2.14)	.73 (0.93)	7.90 (3.81)
	FL**	2.37 (1.04)	1.53 (0.59)	.43 (0.24)	.50 (0.61)	1.99 (1.36)	.40 (0.70)	7.23 (2.79)
Pass 2	FL*	1.42 (0.80)	1.83 (1.03)	.53 (0.50)	.68 (0.63)	.93 (0.92)	.19 (0.36)	5.57 (2.50)
	PDL**	1.88 (0.73)	1.36 (0.88)	.38 (0.18)	.26 (0.31)	1.24 (0.99	.20 (0.35	5.31 (1.73)

* Group 1 subjects used a PDL for pass 1, a FL for pass 2.

** Group 2 subjects used a FL for pass 1, a PDL for pass 2.

Quality and Properties of Designs

Unlike an implemented program, which can, at least conceivably, be evaluated by subjecting it to objective test, the quality of unimplemented software designs can be assessed only by subjective evaluation. Furthermore, in this particular study, the use of different documentation methods precluded "blind" ratings. Clearly, the individual evaluating the design would be aware of the experimental condition in which the design was generated. Mapping all the designs into a common notation was considered, but was not feasible without at least reducing the "naturalness" of the transformed designs.

Within these constraints, quality ratings were obtained quite carefully, with considerable attention by the rater (JRV) to the possibility of experimenter bias. A priori grading criteria were developed (Figure 2) and applied to the designs. The resulting design quality scores are summarized in Table 3. Because the pass 1 and pass 2 assembler designs were dissimilar in function, the grading categories do not correspond. The total scores, however, are intended to be comparable in significance, and are on a scale from zero to 100, where a higher score represents better performance.

Although it would have been preferable to use additional raters, both to improve rating reliability and to provide a measure of that reliability, the only other available person with the requisite qualifications was the course instructor. Since it was necessary to separate the administration of the experiment from the course being taught, the instructor could not be used as a rater. Since this task requires a large amount of time and resources, additional raters were not sought.

Figure 3 provides a graphical illustration of the effect of the experimental conditions on total design score. A corresponding analysis

Pass 1 grading categories
100 point total
1) 20 points
Location counter logic
2) 30 points
Error detection
Invalid operation codes
Labels - required
Multiple label definitions
Out of range location counter
Premature end of file
3) 20 points
Symbol table entry for defined symbols
4) 15 points
Output for input to pass 2
5) 15 points
General logic

Pass 2 grading categories
100 point total
1) 20 points
Error detection
Undefined Symbols
Out of range absolute addresses
and constants
Premature end of file
2) 25 points
Object word creation
3) 40 points
Text buffer management
Initialization
Conditions for buffer output
Logic for building buffer
Trailer record
4) 15 points
General logic

Note: "General logic" categories are intended for factors which do not fall under the other specified categories.

Figure 2. Grading Criteria for Assembler Designs

Table 3. Means and (Standard Deviations)
of Design Quality Ratings

Pass 1

Group	Category (see Figure 2) 1	2	3	4	5	Total
PDL	13.3 (6.41)	18.7 (6.85)	14.2 (6.53)	10.5 (3.03)	10.8 (3.12)	67.5 (22.71)
FL	17.6 (5.25)	18.9 (5.20)	13.1 (5.57)	10.8 (3.12)	11.6 (1.84)	72.0 (10.68)

Pass 2

Group	Category 1	2	3	4	Total
FL	9.2 (5.07)	12.7 (7.38)	17.6 (12.76)	7.7 (4.16)	47.2 (27.76)
PDL	12.3 (4.57)	19.0 (3.83)	30.5 (5.58)	11.4 (1.35)	73.2 (11.25)

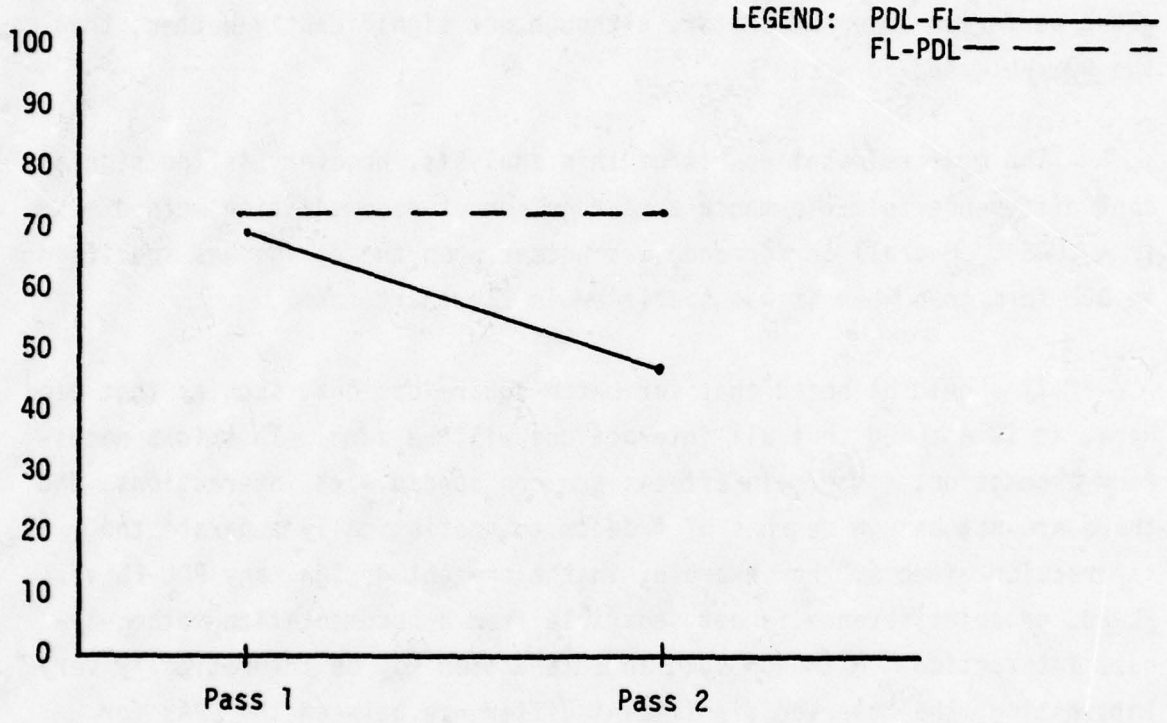


Figure 3. Mean Total Design Score, by Group

of variance is shown in Table 4. Rated performance is significantly lower on pass 2 than on pass 1 ($p < .05$). These design tasks are dissimilar, although related, and we had no particular expectation with respect to their relative difficulty. The grading categories and criteria are necessarily different, and any temporal effects are confounded with the pass 1 - pass 2 difference. As one might expect, given the difference in ability between the groups (more precisely, the difference in GPAs), the FL-PDL group performed somewhat better, although not significantly better, than the PDL-FL group ($p < .10$).

The most relevant result of this analysis, however, is the significant difference in performance as a function of documentation method ($p < .025$). Overall performance was better when the design was specified in PDL form than when it was specified in flowchart form.

It should be noted that for Latin square designs, such as that used here, it is assumed that all interactions will be zero. This is a necessary assumption, since main effects are confounded with interactions, and there are not enough degrees of freedom to statistically separate the interaction effects. For example, in the present design, any PDL-FL vs. FL-PDL group difference is not separable from a documentation-method-by-pass interaction. Although such an interaction may be theoretically very interesting, the observed significant difference between the GPAs for these two groups leads us to conclude that this is, in fact, a group difference and not an interaction.

In either case, however, whether this is interpreted as a group difference or an interaction, its presence reduces the probability of observing significant effects on the other variables, including documentation method. That is, this results in a conservative analysis and the actual significance levels obtained for documentation method are somewhat lower than those reported, although it cannot be determined how much lower (cf. McNemar, 1969, pp. 383-387).

Table 4. Analysis of Variance of Total Design Scores

	Factor	df	Mean Square	F	
Between Groups:	Experimental Group	1	2325	3.82	($p < .10$)
	Subjects w/i Groups	18	10952		
Within Groups:	Assembler Pass	1	912	5.89	($p < .05$)
	Documentation Method	1	1156	7.47	($p < .025$)
	Residual	18	155		

Based on the total design scores, a single design was selected from each of the four groups for inclusion in this report. These designs constitute Appendix P, and represent the most nearly "average" design (in terms of design score only) from each condition.

The designs were analyzed for differences in style, level of detail, and types of constructs used which might be related to documentation method. Factors analyzed included the number of modules used in the design, the number of Boolean expressions (conditional tests), subroutine call operations, and other executable operations used, and the extent and type of abbreviation used in the design. The results, summarized in Table 5, are discussed in the following paragraphs.

PDL and flowchart conditions differed somewhat with respect to the number of modules defined in the designs. Subjects tended to use more modules in designs produced in PDL form than in designs produced in flowchart form ($F(1,18) = 3.436$, $p < .10$).

It seems possible with different documentation methods that designers might vary the kinds of information contained in the design and the constructs used to express it. Such differences are very difficult to detect in an objective manner. Several manual scoring algorithms were developed and rejected for measuring usage of particular design constructs (number of conditional expressions, transfers of control, call statements, etc.) before adopting one which seemed meaningful, unbiased, and relatively unambiguous when applied to either design documentation type. By way of illustrating the problems encountered, consider transfer of control operations. In the context of either documentation type considered in isolation, it is relatively easy to count transfer of control operations. In a PDL, one might count GOTO and RETURN statements, while in flowcharts one might count RETURN operations and arcs emanating from decision diamonds. The difficulty is that these measures are not equivalent, across documentation methods, in any meaningful sense. Many

transfer-of-control operations which are represented explicitly in a flowchart may be represented implicitly in a PDL (as in an IF...THEN statement, DO WHILE, procedure block with an END but no RETURN, etc.). There are other complicating considerations too detailed to warrant discussion here.

The approach actually employed was to count the conditional (Boolean) expressions, call operations, and other executable operations in each design. Conditional expressions are easily recognized in decision diamonds and DO WHILE blocks in flowcharts and in IF...THEN, DO WHILE, and similar constructs in a PDL. The number of conditional expressions is probably related to the number of transfers of control required to implement the design, although it may clearly be affected by the level of detail of the design. The number of subroutine call operations might be related to modular structure, but may also be affected by the provision of an explicit called-subroutine convention in the flowchart method used. The three categories, taken together, include all the executable operations of each design, but exclude such purely structural statements as noniterative DO, END statements, etc. The sum of the three counts is the best metric we were able to devise for measuring the level of detail of the designs.

Table 5 contains the results of this analysis. The principal difference observed was a highly significant tendency for PDL designs to contain more conditional expressions than flowchart designs ($F(1,18) = 11.44$, $p < .005$). Although the PDL-FL and FL-PDL groups differed in their frequency of use of call operations ($F(1,18) = 7.21$, $p < .05$) and other executable operations ($F(1.18) = 4.54$, $p < .05$), frequency of use of these constructs was not related to documentation type. PDL-FL and FL-PDL groups also differed with respect to total number of constructs used ($F(1,18) = 6.28$, $p < .05$), but on that measure there was also a documentation type difference ($F(1,18) = 4.30$, $p < .06$).

Table 5. Means and (in Parentheses) Standard Deviations of "Design Style" and Level-of-Detail Measures

Factor	Pass 1		Pass 2	
	FL	PDL	PDL	FL
Number of modules	3.50 (2.68)	3.40 (1.58)	4.80 (3.12)	2.20 (1.62)
Constructs used				
Conditional expressions	15.8 (5.96)	16.3 (8.22)	19.0 (6.04)	10.9 (6.38)
Call operations	9.50 (7.62)	3.60 (3.41)	8.20 (6.30)	3.00 (2.67)
Other executable operations	37.4 (8.04)	31.7 (12.82)	40.5 (13.97)	28.2 (9.70)
Total	62.7 (17.9)	51.8 (18.3)	67.7 (20.0)	42.1 (16.1)
Number of variable names abbreviated	7.70 (6.00)	2.90 (3.11)	6.40 (4.86)	4.80 (3.85)

The latter difference, which suggests a trend toward more detail in the PDL designs than in the FL designs, is partially, but not entirely, attributable to the difference already noted in usage of conditional expressions.

Detailed study of the designs produced with the two documentation methods leads to a somewhat clearer picture, and suggests that the differences in design quality scores and level of detail are related. Designs expressed in flowchart form tended to contain greater detail in such areas as initialization than those expressed in a PDL, but much less algorithmic detail. For example, a flowchart design might specify "X=0", "Y=0", "Z=1", while a PDL design might merely say "INITIALIZE X, Y,Z". The flowchart design might only say "COMPUTE LOCATION COUNTER ADDRESS", though, while the PDL design describes the actual procedure for this computation. Somehow, these two documentation media seem to lead to a different emphasis on the part of the designer. It is important to note that this observation holds true even though the same subjects used both methods, and is not a result of the group difference.

Designs expressed in flowchart form exhibited a marked tendency toward the use of abbreviated variable names and other space-conserving practices. This tendency is probably related to the need to compress a lot of information into the rather confined space available in flowchart blocks, and was observed to a much lesser degree in PDL designs. In pass 1, flowchart subjects abbreviated an average of 7.7 unique variable names in their designs, while PDL designs contained an average of only 2.9 abbreviated symbols. This difference is significant ($F(15) = 2.246$, $p < .05$). Data from pass 2 are ambiguous, since subjects tended to continue usage of the abbreviations developed in pass 1, even though they switched documentation methods.

It is informative to consider also the way in which subjects abbreviated variable names. Table 6 lists the abbreviations used, and the number of subjects, by documentation type, whose designs contained

Abbreviation	Number of Designs Using Abbreviation	
	PDL	FL
ERROR_MSG	1	
HEX_INSTR	1	
HEX_VALUE	1	
INPUT_LOC	1	
LOC_CTR	1	
MULT_SYMB	1	
NUM_OPCODE	1	
P	1	
SC	1	
SYM	1	
SYM_OPCODE	1	
VAL	1	
E&C	1	
*	1	
ADDR	4	8
ERR	1	4
HEX_IMAGE	1	1
HEX_LOC	1	1
IAC	1	1
LOC	3	5
LC	1	1
MSIZ	1	1
OP	1	6
ADDR1		1
ADDR2		1
N		1
L		1
LCTR		1
HEX (. . .)		3
HOP		1
ADR		1
HADD		1
ADD		1
HLOC		1
LCNT		1
INCR		1
ILOC		1
HL		1
IL		1
INSTR		3
J		1
K		1
I1		1
K1		1
K2		1
K3		1
K4		1
TABLE_VAL		1
OPER		1
I		3
CNTL		2
IMAGE_LOC		2
IMAGE_INSTR		1
HEXLOC		1
CC		1
NUM		1
INST		1
LOC_CNTR		1
NLOC		1
HLOC		1
Total	28	73

Table 6. Summary of Abbreviations Used in Pass 1 Designs

these abbreviations. There appears to be a tendency for flowchart designs to contain a larger proportion of abbreviated variable names whose significance is not clear from the names themselves (i.e., "non-mnemonic" variable names like HL, I1, etc.).

Another observation by the rater was that the selection of appropriate data structures by the designers was very strongly related to their overall design scores. The rater divided the subjects into two groups (best half, worst half) on the basis of data structure usage. The average overall design scores (both passes) for these groups, 155.6 and 104.0, respectively, differ significantly ($t(18) = 4.24$, $p < .001$). Clearly, this is not an a priori hypothesis, nor are these two subjective assessments (ratings of data structure usage, overall design quality) in any sense independent. Even if they were, it would not be clear to what degree data structure usage is a causative factor in design performance and to what degree both are just similarly affected by subject differences in ability, background, etc. The observation is interesting, though, and suggests an area for further investigation.

SUBJECTIVE RATINGS OF DOCUMENTATION METHODS

A statistical analysis of the Flowchart-PDL comparative questionnaire (Appendix J) revealed several significant preferences or opinions, by the subjects, with respect to the two documentation methods. Each response to items 1-27 and 29 was encoded on a scale of 1-8, where a score of 1 represents an extreme preference for flowcharts and 8 represents an extreme preference for PDLs. Since the ratings were obtained after all subjects had been exposed to the use of both documentation methods, the PDL-FL group and the FL-PDL group were not expected to differ significantly in their responses to the questionnaire. This was verified by a series of t-tests, comparing the responses of the two groups on each item. No overall pattern of differences emerged, and the number of

significant group differences was at the chance level. This allowed us to combine the groups for the purpose of analyzing overall preferences. Means and standard deviations of the ratings by all 20 subjects are shown in Table 7.

The questionnaire items were included because they might generate some insight into designer and programmer perceptions of the documentation methods. In most cases, no clear basis existed for an a priori hypothesis concerning the direction, or even the occurrence, of a preference. In eight cases, though, an a priori hypothesis was made, and the direction of expected preference is indicated in parentheses in the corresponding row of Table 7.

Statistical analyses indicated that four of these items showed at least a moderate ($p < .10$) deviation from 4.50, the value expected if subjects have no preference or opinion. These four items are among the eight on which a priori hypotheses were stated, and are in the expected direction. Subjects felt that PDLs were preferable to flowcharts with respect to ease of implementation in a programming language ($t(19) = 2.76$, $p < .02$), ease of documentation ($t(19) = 2.68$, $p < .02$), encouragement of "good" design practices ($t(19) = 2.45$, $p < .05$), and ease of modification of documentation ($t(19) = 1.77$, $p < .10$). On overall preference measures, the subjects exhibited mild (non-significant) preferences for PDLs for use by the designer, but indicated an expectation that flowcharts are preferable from the viewpoint of the programmer. It should be noted that this questionnaire administration preceded the programming task. This is the only item (of eight) on which the direction of preference was opposite that predicted a priori.

Questionnaire items 16-27 asked how PDLs and flowcharts compare with respect to helping the designer notice problems with the design. Question 28 was intended to determine whether the students understood

the design properties referred to in those questions. Seven subjects reported that questions 16-27 were easy to answer, eight students indicated that they were difficult to answer because the subjects saw no difference between the documentation methods, and five indicated difficulty understanding the design errors and problems referred to in the questions.

Table 7. Means and (in Parentheses) Standard Deviations of Questionnaire Ratings after Design Phase

Item	Mean (Standard Deviation)
1. Ease of use by designer (PDL)**	5.00 (1.95)
2. Ease of use by programmer (PDL)	4.15 (1.79)
3. Helping designer understand problem	4.35 (1.79)
4. Encouragement of "good" design practices (PDL)	5.40*(1.64)
5. Ease of documentation (PDL)	5.50*(1.67)
6. Ease of modification of documentation (PDL)	5.05*(1.39)
7. Ease of modification of design	4.00 (1.70)
8. Expression of control flow (FL)	3.80 (2.24)
9. Expression of data flow	4.15 (1.87)
10. Comprehensibility of design	4.95 (1.93)
11. Ease of reading	4.65 (1.95)
12. Understanding of control flow	4.35 (1.98)
13. Understanding of data flow	4.40 (1.79)
14. Ease of implementation in programming language (PDL)	6.00*(1.62)
15. Detection of design problems	4.30 (1.84)
16. Detection of redundant operations	4.25 (1.86)
17. Noticing that module is too big	4.95 (1.50)
18. Noticing that module is too complex	4.45 (1.67)
19. Module has too many or too few submodules	4.55 (1.54)
20. Module is not cohesive	5.03 (1.44)
21. Definition is not detailed enough	4.35 (1.57)
22. Definition has too much detail	4.35 (1.53)
23. Detection of incomplete module	4.30 (1.75)
24. Modules tightly coupled	4.55 (1.70)
25. Design too complex	4.05 (1.67)
26. Data Fragmentation	4.30 (1.22)
27. Module is trivial	4.80 (1.70)
29. Overall impression (PDL)	4.55 (1.64)

* Significantly different from chance (4.5). See text.

** Indicates an a priori hypothesis that PDL would be preferred.

IMPLEMENTATION PHASE

In the implementation phase, each group was exposed to only one documentation method. If the two groups of subjects had been found to differ with respect to relevant ability or background variables, as occurred in the design phase, that difference would have been confounded with the effect of documentation type. However, the two implementation-phase experimental groups (FL, PDL) did not differ significantly on any known background or ability factors. Any performance differences in the implementation phase may therefore be attributed to the effects of documentation type.

The subjects from each design phase group were distributed equally into FL and PDL implementation phase groups. Experience during the design phase is therefore counterbalanced in the implementation phase. Factorial analyses of several key implementation-phase variables (e.g., scores on design comprehension test, grades on implementation project, properties of implemented program) reveals no pattern of interaction between design and implementation conditions. Therefore, implementation-phase results will be reported only by implementation condition (FL or PDL). Because of missing data and other indications by subjects that they were unable to produce reliable estimates of their time expenditure during the implementation phase, these data will not be reported.

Comprehension of Simulator Design

Observed effects of documentation type on design comprehension were small and not statistically significant. Table 8 shows the results of the multiple-choice portion of the design comprehension test administered after 45 minutes' study of the design (first test) and again at the end of the entire implementation phase (second test). It is important to note that results reported for the first test are based on all 20 subjects, while those for the second test exclude subjects who did not complete the study.

Table 8. Results of Multiple-Choice Portion of Design Comprehension Test

Category	Items	First Test		Second Test	
		FL	PDL	FL	PDL
Total Score	All	17.0 (3.86)	17.1 (5.72)	21.9 (3.48)	20.6 (4.90)
Comprehension of Micro Nova	5, 14, 26	1.80 (.789)	2.50 (.707)	2.75 (.463)	2.75 (.463)
Distinction between Micro-Nova and Simulator	11, 12, 24	1.90 (.738)	1.80 (.919)	2.00 (.535)	1.63 (.916)
Debugging of Simulator	2, 10, 20, 29	1.80 (1.03)	2.00 (1.25)	2.88 (.641)	2.13 (1.13)
Comprehension of Impact of Changes	6, 9, 19, 27	2.20 (.919)	1.50 (1.18)	2.63 (1.06)	3.13 (.991)
Comprehension of General Structure	4, 8, 15, 16, 21, 23, 25, 28	4.10 (.876)	4.60 (1.90)	4.75 (1.04)	5.00 (1.69)
Comprehension of Procedural Detail	1, 3, 7, 13, 17, 18, 22, 30	5.20 (2.26)	4.70 (2.26)	6.50 (1.77)	6.00 (1.60)

On the total score, and most of the subscores, statistically significant improvement was observed from the first test to the second, as might be expected.

On the first test, students using a PDL scored somewhat better than FL subjects on comprehension of the MicroNova ($t(18) = 2.090$, $p < .06$); this difference disappeared entirely on the second test. It should be noted that this test category (as well as the next category, Distinction between MicroNova and Simulator) was intended as a measure of background knowledge, rather than of information contained in the design itself. It is possible, however, that the PDL design somehow aided rehearsal of this information, or otherwise reinforced it more than did the flowchart design. The MicroNova information is derivable from the design; it was considered background information primarily because the student was expected to know it already.

Repeated-measures analyses of variance on the total score and each subscore (using data for the 16 subjects who completed the study) found no flowchart-PDL differences which were consistent across both test administrations.

As a priori hypothesis relating to subscores was that flowcharts might aid comprehension of procedural detail, while PDLs might improve comprehension of the general structure and function of the design. The data indicate mild, but statistically insignificant, tendencies in the expected directions.

No significant differences were observed between PDL and FL subjects with respect to the number of modules they were able to correctly identify from the design, or with respect to the number of calling-module/called-module relationships correctly identified. These are probably not sensitive measures of design comprehension.

Quality of Implementations

We originally intended to evaluate the simulator programs by the method of Youngs (1974), in which the final and all earlier runs are analyzed for errors in reverse chronological order. This method is very effective in detecting a variety of error types, by the programmer, including clerical, snytactic, and various types of conceptual error. However, the number of runs involved was much larger than anticipated (1192), and such an approach would have been an overwhelming task, particularly given the size of the programs (average length, 1030 source statements). As a result, only the final, completed programs were evaluated.

The completed simulator programs were evaluated by an expert rater (JRV) on the basis of the a priori grading criteria shown in Figure 4. These ratings were done on the basis of: (1) the finished source programs (as compiled, with symbol table, cross-reference table, etc.), (2) results of a fairly extensive test program, and (3) reference to earlier simulator development runs as needed. The results of this evaluation are shown in Table 9. Statistical analysis indicates that there are no significant differences between flowchart and PDL groups on total score or on any of the subscores. No particular classes of implementation error were noted which seemed to be attributable to design documentation type.

Appendix Q contains the program which was most nearly average in quality. This particular implementation was developed from a flowchart design.

The final programs were also evaluated with respect to several simple measures of programming style or programming practice. No differences were observed between FL and PDL subjects with respect to number of unique variable names, total number of variable name references,

Implementation Evaluation Categories

Category 1 (50 points)

Location counter modification and referencing logic
- Incrementation associated with fetching
- Conditional modification based on skip indicator
- Relative addressing
- Jumps and subroutine calls
- Traps and interrupts
- Subroutine returns
- Two word block I/O instructions
- Setting and testing active trace location

Category 2 (50 points)

Effective address computation and usage
- Indexing and displacement addressing
- Indirect addressing including chaining and auto indexing
- Usage of effective addresses
  - Interrupts and traps
  - I/O
  - Memory reference instructions

Category 3 (15 points)

Dump and trace control logic

Category 4 (25 points)

Interrupt detection and generation including real time clock management and interrupt enable/disable logic.

Category 5 (15 points)

Instruction class resolution and invocation.

Category 6 (25 points)

Instruction simulation excluding logic covered by previous categories.

Total - 180 points

Figure 4. Grading Criteria for Implemented Simulators

Table 9. Means and (Standard Deviations)
of Implementation Quality Ratings

Group	Category (see Figure 4) 1	2	3	4	5	6	Total
FL	34.38 (14.97)	43.25 (7.83)	12.63 (3.74)	21.13 (4.79)	15.00 (0)	20.50 (4.50)	146.88 (28.42)
PDL	33.75 (14.40)	43.25 (6.45)	12.75 (4.17)	19.88 (5.19)	13.75 (1.75)	20.63 (4.27)	144.00 (28.66)

or number of usages of each variable. The flowchart group used 26% more "GO TO" statements than did the PDL group (means were 36.75 and 29.13, respectively), and this difference remained when adjustments were made from program length. However, the variances were very large, and this mean difference does not approach statistical significance.

The FL and PDL groups also exhibited differences in average program length and in the number of runs made during program development. As in the case of "GO TO" statements, the mean differences are fairly large, but are not statistically significant. The FL and PDL subjects averaged 63 and 86 runs, respectively, but this difference is not significant ($F(1, 14) = 2.272$, $p > .10$). The average program length (number of source statements) for FL and PDL subjects, respectively, was 1136 and 924, a non-significant difference ($F(1, 14) = 1.893$, $p > .10$).

Subjective Ratings of Documentation Methods

Results of the post-implementation comparative questionnaire are shown in Table 10. Overall, subjects indicated significant preference for PDL over flowchart documentation ($t(15) = 2.15$, $p < .05$). Specifically, significant PDL preferences were noted with respect to ease of documentation ($t(15) = 4.04$, $p < .01$), ease of modification of documentation ($t(14) = 2.40$, $p < .05$), ease of implementation in a programming language ($t(15) = 3.95$, $p < .01$), and ability to detect an insufficiently detailed module in the design ($t(15) = 2.18$, $p < .05$). Students tended also to prefer PDLs with respect to encouragement of "good" design practices ($t(15) = 2.06$, $p < .10$). On questionnaire items involving *a priori* hypotheses, eight out of eight preferences were in the hypothesized direction, with four significant at the $P < .05$ level (and one trend at $P < .10$).

The subjective ratings may have been biased by a lecture delivered at OSU by well-known computer scientist Daniel McCracken shortly before

the questionnaire was administered. In the lecture, McCracken extolled the virtues of PDLs at some length. With respect to overall preference (item 29), subjects attending the lecture produced a mean rating of 6.25, as compared with 4.75 for those not attending the lecture. Variances were large, however (standard deviations were 1.28 and 2.12, respectively), and this mean difference does not achieve statistical significance ($t(13) = 1.71$, $p > .10$). The cause and effect are not entirely clear, as the more highly motivated students were probably more prone to attend the lecture. Nonetheless, this biasing factor must be considered in interpreting the results.

The results shown in Table 10 actually represent two groups (PDL, FL) which were not entirely in agreement. As one might expect, subjects' preferences tended to be influenced by the documentation method they actually used during the implementation phase. On most questionnaire items, PDL subjects indicated stronger preference for PDLs than did FL subjects. These group differences were only significant on a few items, however, on item 12, understanding of control flow, mean ratings were 5.75 and 3.63, respectively ($t(14) = 2.26$, $p < .05$). On item 15, detection of design problems, means were 5.88 and 4.13 ($t(14) = 3.30$, $p < .01$).

Immediately after the initial study of the designs and administration of the first comprehension test, several students using flowcharts commented spontaneously that their impression of the relative readability of the two documentation methods had changed considerably as a result of their attempt to understand the flowchart design. This was reflected later in their responses on item 11 of the questionnaire. Between administrations of the questionnaire, the ratings on this item, by students using flowcharts, changed by an average of 1.38 points in favor of PDLs (by a difference test, ($t(7) = 4.88$, $p < .01$). The responses of both PDL and FL subjects also changed an average of .94 points in favor of PDLs on item 21, detection of insufficiently detailed module ($t(15) = 4.04$,

Table 10. Means and (Standard Deviations) of Questionnaire Ratings after Implementation Phase

Item	Mean (Standard Deviation)
1. Ease of use by designer (PDL)**	5.25 (1.77)
2. Ease of use by programmer (PDL)	5.13 (2.06)
3. Helping designer understand problem	5.0 (1.79)
4. Encouragement of "good" design practices (PDL)	5.56*(2.06)
5. Ease of documentation (PDL)	5.88*(1.36)
6. Ease of modification of documentation (PDL)	5.4 *(1.45)
7. Ease of modification of design	4.73 (1.87)
8. Expression of control flow (FL)	3.80 (1.86)
9. Expression of data flow	4.0 (1.52)
10. Comprehensibility of design	5.0 (1.96)
11. Ease of reading	5.0 (1.78)
12. Understanding of control flow	4.69 (2.12)
13. Understanding of data flow	4.47 (1.46)
14. Ease of implementation in programming language (PDL)	6.25*(1.77)
15. Detection of design problems	5.0 (1.37)
16. Detection of redundant operations	4.38 (1.45)
17. Noticing that module is too big	4.75 (1.44)
18. Noticing that module is too complex	4.38 (1.50)
19. Module has too many or too few submodules	4.81 (1.47)
20. Module is not cohesive	4.63 (1.26)
21. Definition is not detailed enough	5.31*(1.49)
22. Definition has too much detail	4.81 (1.42)
23. Detection of incomplete module	4.63 (1.41)
24. Modules tightly coupled	4.33 (1.40)
25. Design too complex	4.25 (1.53)
26. Data Fragmentation	4.20 (1.21)
27. Module is trivial	4.81 (1.38)
29. Overall impression (PDL)	5.50*(1.86)

* Significantly different from chance (4.5). See text.

** Indicates an a priori hypothesis that PDL would be preferred

$p < .01$). Student comments seem to indicate that these rating changes reflect problems encountered by the students during the implementation task.

DISCUSSION

The results, indicate that designs produced in PDL form had higher overall quality than those produced in flowchart form. PDL designs were more detailed, particularly with respect to algorithmic detail, and involved much less abbreviation of variable names than flowchart designs. When designs were presented to subjects in PDL or flowchart form, however, no pattern of short or long-term differences was detected on measures of design comprehension. No significant differences were detected in the quality or other properties of the programs written as implementations of the designs. Subjective ratings indicated a mild overall preference for PDLs, as well as several specific criteria on which students found PDLs preferable to flowcharts.

The measures of design quality which were used to assess subject-generated designs can be criticized as too subjective. However, it is not clear that any more objective approach is available, given our current level of understanding of the software design process. In any event, the differences in logical quality of the designs were fairly large, favored PDLs fairly consistently, and were generally consistent with other, more objective observations concerning the level of detail and readability of the produced designs.

The observed difference in level of detail between flowchart and PDL designs is particularly interesting. The fact that PDL designs contained 32% more conditional expressions than did flowchart designs suggests that they contain a greater degree of algorithmic or procedural detail. This may, in fact, be a significant factor in their higher assessed quality. The finding that PDL designs contained 44% more modules than did flowchart designs is also suggestive of greater detail, although it is possible that PDLs simply encourage modularization. Notice, however, that a higher level of procedural detail implies the

existence of a larger number of non-conditional expressions, as well as a larger number of conditional expressions. Yet, a 32% difference in conditional expressions was accompanied by only a 10% difference in "other executable operations". Examination of the designs suggests that the flowchart designs contain more detailed information concerning some aspect of the design which is not associated with condition-testing. A particularly prevalent difference concerns initialization of variables. While PDL designs tend to specify initialization with a single ambiguous statement ("INITIALIZE VARIABLES", or something similar), flowchart designs tend to contain explicit, unambiguous specifications of each initialization operation (e.g., "SET X=0, "SET Y=0", SET Z=1").

Informal program design languages are similar to natural language to a much greater degree than are flowcharts. Studies of query formulation by non-programmers (Gould, et al 1976; Miller and Becker, 1974) have observed the frequent use of procedurally ambiguous statements which require interpretation by the reader. The use of such statements as "INITIALIZE VARIABLES" in PDL designs may be related to those observations. In using natural language, people apparently tend to assume that the recipient of the communication will possess sufficient semantic information to correctly interpret such imprecise statements. The flowchart "language", on the other hand, is much more formal and may encourage the use of detailed, unambiguous individual statements.

PDL and flowchart usage differed not only in overall level of detail, but also in the aspects of the designs selected for detailed specification. Many would argue that it is the algorithmic detail that is most important in the specification of these designs, and that details in initialization and similar information can be provided by the programmer. If true, the data suggests that the designer's use of PDLs encourages more detailed specification of relevant aspects of the design than does his use of flowcharts.

When subjects were presented with logically equivalent designs expressed either in PDL or flowchart form, no differences were observed in either "short-term" (45 minutes) or long-term comprehension of the designs. The lack of a short-term comprehension difference seems inconsistent with the findings of Wright and Reid (1973), whose subjects remembered a procedure better when expressed in a form somewhat similar to a PDL than when expressed in flowchart-like form. It should be noted, however, that comprehension of this simulator design is quite a formidable task when compared with their simple algorithms, which typically involved only three binary decisions. It is questionable whether 45 minutes' exposure to the simulator design represent a "short-term" presentation in the same sense as the much shorter exposure to simple algorithms in the Wright and Reid study. Furthermore, their recall measure (execution of algorithm from memory) is probably a more sensitive performance measure than our design comprehension test. Finally, the specialized background of our programmer-subjects may have led them to develop more efficient methods for processing flowchart information and encoding it in memory.

This experiment is a conservative test of the relative comprehensibility of the two documentation methods. In order to investigate differences in documentation method in isolation from other factors, it was necessary to carefully control the way in which design statements were worded. Wherever possible, complete statements were expressed identically in PDL and flowchart designs. This resulted in flowcharts which were more "wordy" than usual, but which were locally (i.e., at the level of individual statements) quite readable. The primary issue addressed in the experiment was the effect of documentation type on global comprehensibility.

The results suggest that flowcharts and PDLs are equally comprehensible if they contain similar, readable, unabbreviated statements.

If used well, either method may allow effective communication with the programmer. In actual practice, however, the use of flowcharts appears to encourage the use of highly abbreviated symbols and other space-saving techniques. Although the present study has not attempted to determine whether or not such abbreviations adversely affect the comprehensibility of designs, it would be surprising if they did not do so, at least in the short term. Thus, in actual practice flowchart designs may be less comprehensible because of the medium's effect on the behavior of the designer, rather than its effect on the programmer.

Design documentation method was not found to affect the quality or style of implementation by the programmer. It might be suggested that a good programmer should be able to understand a good design expressed in any reasonable form. In fact, such a suggestion was made by several of the students who participated in the study. The issue may not be that simple, however. The fact that the flowchart and PDL designs were logically equivalent does not imply that they were psychologically equivalent. If the programmer's perception of the design is altered by the form in which the design is presented -- for example, by causing him to attend more to control structure, or more to modularization -- he may, in effect, solve a different problem. There is evidence of such phenomena in the literature of other task domains (e.g., Simon & Hayes, 1976). There was a no a priori justification for assuming that the peculiar properties of PDLs and flowcharts would not exert such differential effects. Based on the data obtained in this experiment, however, such differential effects appear to be either insignificant in magnitude or of a type to which the analyses employed are insensitive. The latter possibility is certainly not to be ignored.

Subjective preferences, as expressed in responses to both questionnaires, and especially the post-implementation questionnaires, indicate a mild but somewhat consistent preference for PDLs over flowcharts. Subjects also indicated several criteria on which PDLs are particularly to be

preferred, including encouragement of "good" design practices, ease of documentation and of modification of documentation, and ease of implementation in a programming language.

Overall, the results appear to provide a fairly strong case for the use of program design languages, in preference to flowcharts, for the expression of detailed software designs by the designer. As with all studies, it must be kept in mind that this study was done using a particular design and programming task, a particular class of designers and programmers, and a particular experimental setting. The degree to which similar results might be obtained in other software design and programming situations depends in part upon their similarity to the conditions under which this experiment was conducted. Generalization to meaningful software development settings was, of course, the primary consideration in our decision to study the behavior of experienced programmers involved in software development efforts of significant size.

CONCLUSIONS

In the context in which this study was performed, the use of a Program Design Language (PDL) by a software designer, for the development and description of a detailed program design, produced better quality designs than did the use of flowcharts. In particular, the PDL designs involved more algorithmic or procedural detail than those produced using flowcharts. In addition, flowchart designs exhibited considerably more abbreviation and other space-saving practices than did PDL designs, with a possible adverse effect on their readability.

When equivalent, highly readable designs were presented to subjects in both PDL and flowchart form, no pattern of short-term or long-term differences in comprehension of the design was observed. No significant differences were detected in the quality or other properties of programs written as implementations of the designs. Subjective ratings indicated a mild preference for PDLs.

Overall, the results suggest that software design performance and designer-programmer communication might be significantly improved by the adoption of informal Program Design Languages, rather than flowcharts, as a standard documentation method for detailed computer program designs.

REFERENCES

Aron, J. The program development process: The individual programmer. Reading, Massachusetts: Addison-Wesley, 1974.

Atwood, M.E., & Ramsey, H.R. Cognitive structures in the comprehension and memory of computer programs: An investigation of computer program debugging (Technical Report). Alexandria, Virginia: U.S. Army Research Institute, in press.

Boehm, B.W. Software and its impact: A quantitative assessment. Datamation, May 1973, 19(5), 48-59.

Boehm, B.W., & Haile, A.C. Information processing/data automation implications of Air Force command and control requirements in the 1980s (CCIP-85): Executive summary (Rev. Ed.) (Technical Report SAMSO/XRS-71-1. Space and Missile Systems Organization, Los Angeles, CA, February 1972. (NTIS No. AD 742292)

Brooks, F.P., Jr. The mythical man-month: Essays on software engineering. Reading, Massachusetts: Addison-Wesley, 1975.

Craik, F.I.M. A "levels of analysis" view of memory. In P. Pliner, L. Kramers, & T.M. Alloway (Eds.), Communication and affect: Language and thought. New York: Academic Press, 1973.

Craik, F.I.M., & Lockhart, R.S. Levels of processing: A framework for memory research. Journal of Verbal Learning and Verbal Behavior, 1972, 11, 671-684.

Data General Corp. User's manual (programmer's reference): MicroNova computers (Ordering No. 015-000050). Southboro, Massachusetts: Data General Corp., February 1976.

Gould, J.D., Lewis, C., & Becker, C.A. Writing and following procedural, descriptive, and restricted syntax language instructions (Technical Report No. RC-5943). Yorktown Heights, New York: IBM Watson Research Center, April 1976.

Jeffries, R., Polson, P.G., Razran, L., & Atwood, M.E. A process model for missionaries -- cannibals and other river crossing problems. Cognitive Psychology, 1977, 9(4), 412-440.

Judd, D.R. The documentation of computer programs. In Software engineering. Berkshire, England: Infotech Information Ltd., 1972, 411-424.

Kammann, R. The comprehensibility of printed instructions and flowchart alternative. Human Factors, 1975, 17, 183-191.

Kraly, T.M., Naughton, J.J., Smith, R.L., & Tinanoff, N. Structured programming series (Vol. 8): Program design study; Final report (Report No. RADC-TR-74-300-VOL-8). Griffiss AFB, New York: Rome Air Development Center, May 1975. (NTIS No. AD A016415)

McNemar, Q. Psychological statistics (4th Ed.). New York: Wiley, 1969.

Miller, L.A., & Becker, C.A. Programming in natural English (Technical Report No. RC-5137). Yorktown Heights, New York: IBM Watson Research Center, November 1974. (NTIS No. AD A003923)

Ortega, L.H. Structured programming series, Vol. VII, Documentation standards (Technical Report RADC-TR-74-300, Vol. VII). Griffiss AFB, New York: Rome Air Development Center, September 1974.

Paivio, A. Mental imagery in associative learning and memory. Psychological Review, 1969, 76, 241-263.

Scott, R.F., & Simmons, D.B. Programmer productivity and the Delphi technique. Datamation, May 1974, 20(5), 71-73.

Shneiderman, B., Mayer, R., McKay, D., & Heller, P. Experimental investigations of the utility of detailed flowcharts in programming. Communications of the ACM, 1977, 20, 373-381.

Simon, H. A., & Barenfield, M. Information-processing analysis of perceptual processes in problem solving. Psychological Review, 1969, 76, 473-483.

Simon, H.A., & Hayes, J.R. The understanding process: Problem isomorphs. Cognitive Psychology, 1976, 8(2), 165-190.

Van Leer, P. Top-down development using a program design language. IBM Systems Journal, 1976, 2, 155-170.

Weinberg, G.M. The psychology of computer programming. New York: Van Nostrand Reinhold, 1971.

White, W.L. Structured software design. In 8th NTEC/Industry Conference Proceedings. Orlando, Florida: Naval Training Equipment Center, 1975, 249-252.

Wright, P., & Reid, F. Written information: Some alternatives to prose for expressing the outcomes of complex contingencies. Journal of Applied Psychology, 1973, 57, 160-166.

Youngs, E.A. Human errors in programming. International Journal of Man-Machine Studies, 1974, 6, 361-376.

APPENDIX A - SCHEDULE OF EVENTS

Event	Date	Appendices
1. Start of course	18 Jan	
2. Explanation of experiment	18 Jan.	B,C
3. Lecture on PDLs, flowcharts	8 Feb.	D,E
4. Pass 1 design started	8 Feb.	F,G,I
5. Pass 1 collected, pass 2 started	within 5 days of 4	H
6. Pass 2 collected	15 Feb.	
7. PDL vs. FL questionnaire	17 Feb.- due 22 Feb.	J
8. Simulator design distributed	22 Feb.	K,L,O
9. Design comprehension test	1 hr. after 8	N
10. Design revision distributed	24 Feb.	M
11. Simulator programs collected	21 Apr.	
12. Second comprehension test	28 Apr.	N
13. PDL vs. FL questionnaire	28 Apr.	J

APPENDIX B - CONSENT FORMS

INFORMED CONSENT FORM

The purpose of this study is to examine the relative merits of two forms of design specification, flowcharts and program design languages, on the design and implementation of software systems. This study will in no way interfere with the stated objectives of this course. We will observe and record your performance on required course projects; you will not be required to perform significant activities in addition to normal course assignments.

We will observe your performance on two major assignments -- a design task and an implementation task. In addition, you will be given course exams and asked to complete a questionnaire describing your background in computer science. Participants in this study will remain anonymous. The sole purpose of this research is to examine alternative forms of design specification; it is not a test of your intelligence or personality. You are not required to participate in this study, and you may change your mind about participating at any time during the experiment. However, payment or other rewards can only be given to those who complete all experimental requirements. At the conclusion of this study, a complete description of this research will be given to you.

Although course grades will be determined, in part, on the basis of information collected during this experiment, experimental manipulations will not influence grades. Your performance will be compared, for grading purposes, only with that of students under identical experimental conditions. The experimental manipulations are not intended to make your task more difficult or to degrade your performance in any way. There is no reason to believe that the experimental materials that are presented to you will either help or hinder your performance relative to students given other materials. In order to insure an accurate assessment of the experimental manipulations to be performed, we ask that you do not compare materials, problems or solutions with fellow students.

Any questions concerning the procedures of this study can be discussed with Dr. Van Doren. These procedures were designed to be consistent with guidelines established by the Department of Health, Education and Welfare concerning the use of human subjects and with Oklahoma State University policies.

The results of this study are expected to be of considerable importance to the software development community. Previous studies have generally involved undergraduates with little, if any, computer science background and very small projects. The results obtained from such studies are frequently equivocal. The study in which you will participate offers the significant opportunity to observe the performance of well-qualified computer scientists on non-trivial projects. This increases the probability of finding meaningful results that can be used to improve computer science training and practices.

Your cooperation is invaluable and greatly appreciated. Because the study is expected to be interesting and beneficial to you, as well as to the software development community at large, we hope you will want to participate. However, as an additional reward for your participation, you will be paid $25.00 upon successful completion of all experimental requirements.

Dr. H. Rudy Ramsey
SCIENCE APPLICATIONS, INC.
40 Denver Technological Center West
7935 East Prentice Avenue
Englewood, Colorado 80110

I have read and understand the above statement and agree to participate in this study.

_____
Name (please print)

_____
Signature

In analyzing the results of this study, it would be useful to have a record of your previous and current courses and the grades received. All information about grades will be treated as confidential and identified only by a code number. A master list assigning code numbers to names will be maintained only by Dr. Van Doren and will be destroyed as soon as possible after completion of this study.

Subject to the conditions stated above, I agree to the release of the information contained in my academic record.

_____

Name (please print)

_____

Signature

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX C - BACKGROUND QUESTIONNAIRE

Name: _____

INSTRUCTIONS

This questionnaire is designed solely for experimental purposes; your responses will in no way affect your course grade. Since we will be observing your performance on course projects, it would be very useful to us to have a brief description of your background in computer science. We ask that you complete these questions accurately and honestly.

Code:_____

1. List previous computer science courses.

   Undergraduate

   _____

   _____

   _____

   _____

   _____

   Graduate

   _____

   _____

   _____

   _____

   _____

2. Are you enrolled in the computer science program? Yes____ No____
   If no, which program are you enrolled in?_____
   If yes, which graduate option?_____
   How many hours have you completed in this option?_____
   How many graduate hours have you completed in computer science?
   _____

3. List previous degrees received

   _____

   _____

4. When do you expect to graduate from this program and what degree will you receive?

   _____

5. Are you a native speaker of English? Yes____ No____
   If no, how long have you lived in English speaking countries?
   Years_____ Months_____
   Were your previous degrees obtained at universities where English was used? Yes____ No____ Some____
   If no or some, briefly describe your educational background in terms of universities, language, years enrolled and degrees received.

   _____

   _____

6. Briefly describe your experiences, outside of normal course work with the following (list types of experience, length, where received, etc.):

PL/1

Flowcharts

Program Design Languages

Other Design Specification Methods (specify)

Assemblers, Translators, and Compilers

Machine Simulators

Mini/Micro Computers

Date General NOVA Computers

Other Related Experiences

7. Age_____

8. Sex_____

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX D - SOFTWARE DESIGN METHODS HANDOUT

Notes on Software Design Methods (Flowcharts and PDL's)

The material presented is similar but not identical to material contained in:

Structured Programming Series, Vol. 8:
Program Design Study
IBM Federal Systems Center
Gathersburg, Maryland

Produced under U.S. Air Force contract
and available from
National Technical Information Service
Ref. # RADC-TR-74-300

should be design - Fisher mentioned this in class

It is assumed that you are familiar with flowcharting and the conventions in use at Oklahoma State University. It is not assumed that you are familiar with Program Definition Languages (PDL's). PDL's typically have control structures similiar to those available in block structured languages such as PL/1 and ALGOL. However, a PDL is used for software design, not as a programming language. Of course there may be direct correspondence with some features in your programming language just as there may be with flowcharts. PDL's may be informal or formal. The latter have a very formal and strict definition. The former have certain well defined control structures but much of the processing can be specified in an informal way just as flowchart procesing blocks may have varying degrees of precision and formality depending on the level of logic or design.

For purposes of the software design study being conducted in this course an informal PDL will be used. As part of your computer science knowledge you should become aware of PDL's as alternative methods for software design (and documentation as well).

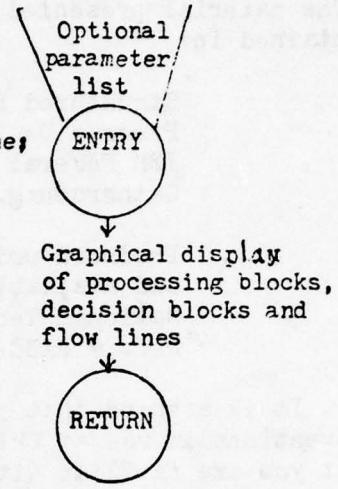
The remainder of the material comprises a parallel presentation of the control structures for the informal PDL and corresponding flowcharting treatments.

Modules or procedures:

PDL

```
Module name: PROCEDURE optional parameter list;
     .
     .
     Sequence of PDL and/or
     English langauge statements
     .
     .
     RETURN TO CALLER;
     .
     .
 END module name;
```

Flowchart

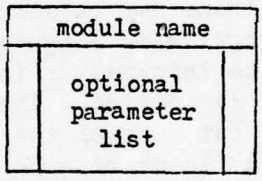


Module invocation:

PDL

```
CALL module name  optional parameter list;
```

Flowchart

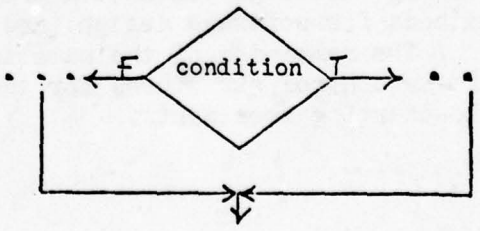


Elementary decision logic:

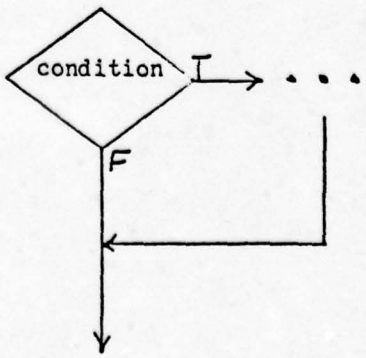
PDL

```
IF  condition  THEN
    English language or PDL statement;
ELSE
    English language or PDL statement;
```

Flowchart




```
IF  condition  Then
    English language or PDL statement;
```

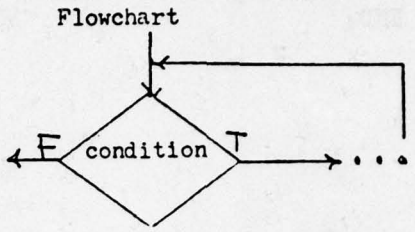


Looping:

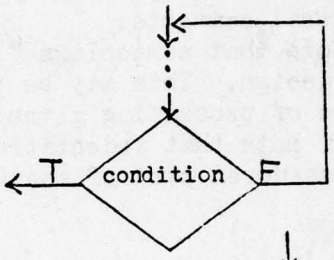
PDL

```
DO WHILE  condition;
   Sequence of English language
   and/or PDL statements;
END;
```

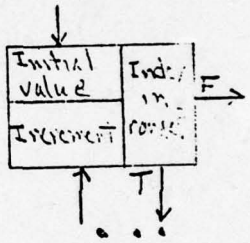
Flowchart




```
DO UNTIL  condition;
   Sequence of English language
   and/or PDL statements;
END;
```




```
DO index = initial value TO final value BY incr.
   Sequence of English language
   and/or PDL statements;
END;
```

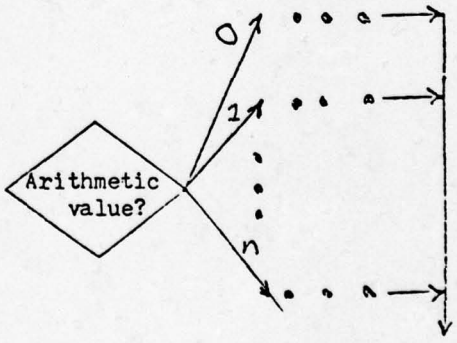


Multiway decision:

PDL

```
DO CASE  arithmetic value;
   CASE0:
      English language statement
      or PDL control statement;
   CASE1:
      English language statement
      or PDL control statement;
   .
   .
   .
   CASEn:
      English language statement
      or PDL control statement;
END CASE;
```

Flowchart



Note1: If the arithmetic value is "out of range" nothingis specified. Some conventions establish a default control path for this. Please observe that a zero arithmetic value is considered in range for this PDL.

Note2: Multiway decisions such as illustrated may easily be simulated inPL/1 with the use of a LABEL vector, the PL/1 equivalent of a "computed GO TO" and a GO TO after each CASE block.

Sequence of design specificiations:

PDL

```
DO;
   Sequence of English language
   and/or PDL statements;
END;
```

Flowchart

Additional comments:

Note that semicolons ";" are used to terminate all statments in a PDL design. This may be particularly important in making clear a seqence of processing givne in the form of English language statements. Further note that indentation schemes for nested PDL specifications are considered part of the PDL method.

APPENDIX E - PDL AND FLOWCHART EXAMPLES

PDL AND FLOWCHART SOFTWARE DESIGN ILLUSTRATIONS

The illustrations are given in terms of a simulator design for the "small computer" discussed in class via APL functional descriptions. There are several items about this design to be stressed:

1) For design illustration reasons it is assumed that an op code of 0000 (binary bits) represents a "HALT" instruction and that op codes of 0011, 0100 and 0111 are invalid. The APL functional description does not include these items. It is further assumed that op codes of 1100 and 1111 represent "TRACE ON" and "TRACE OFF" simulator instructions. These two instructions are not part of the "small computer" instruction set but rather are simulator design provisions for tracing the flow of simulated execution of "small computer" programs.

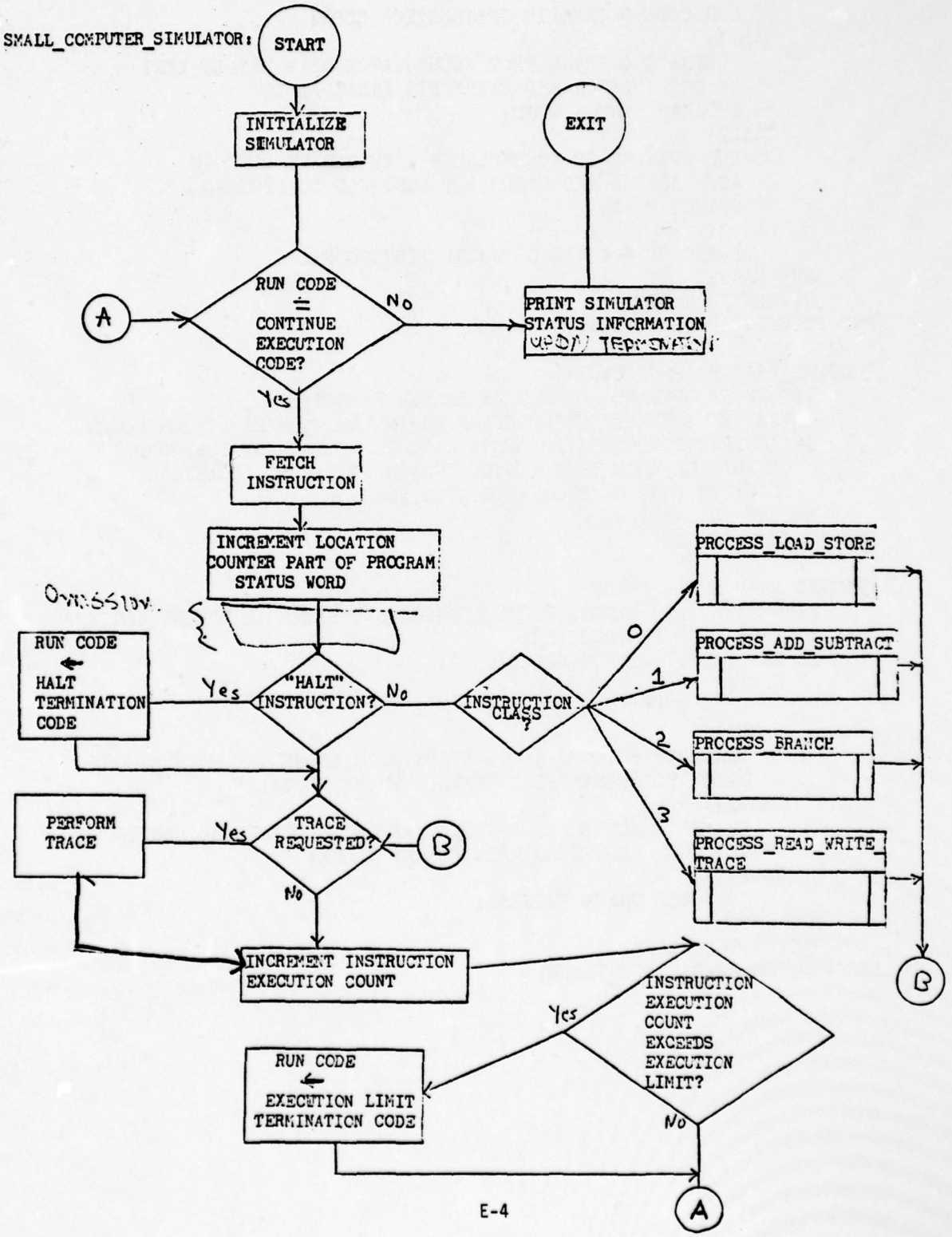
2) The simulator design is a software design which contains features that are not part of the functional characteristics of the simulated machine. Obviously there are some similarities between the simulator design and the APL functional characteristics.

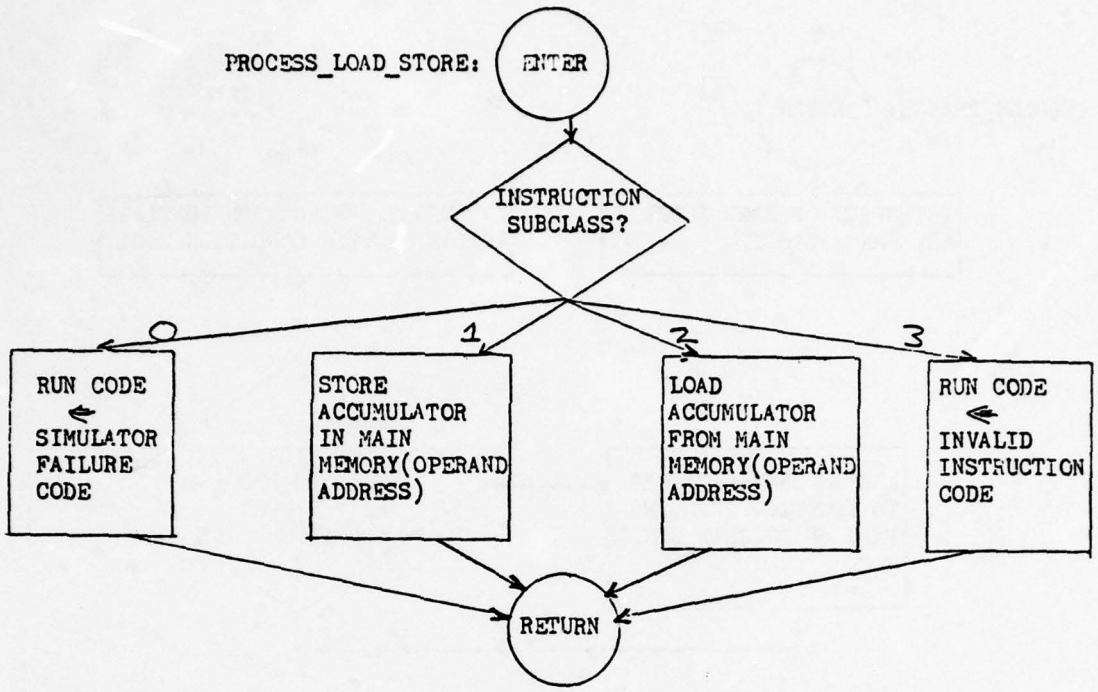
3) Neither of the software designs includes the mathematical precision of the the APL functional specifications. Precise details for simulating certain functional characteristics should be deduced from the APL specifications. Programming details do not require direct correspondence to APL features. You need only assure logical equivalence. The software design shows where the detailed programming for these items belongs in the general scheme of things. The notion of logical equivalence of the implemented program applies as well to software design features as it does to APL description features.

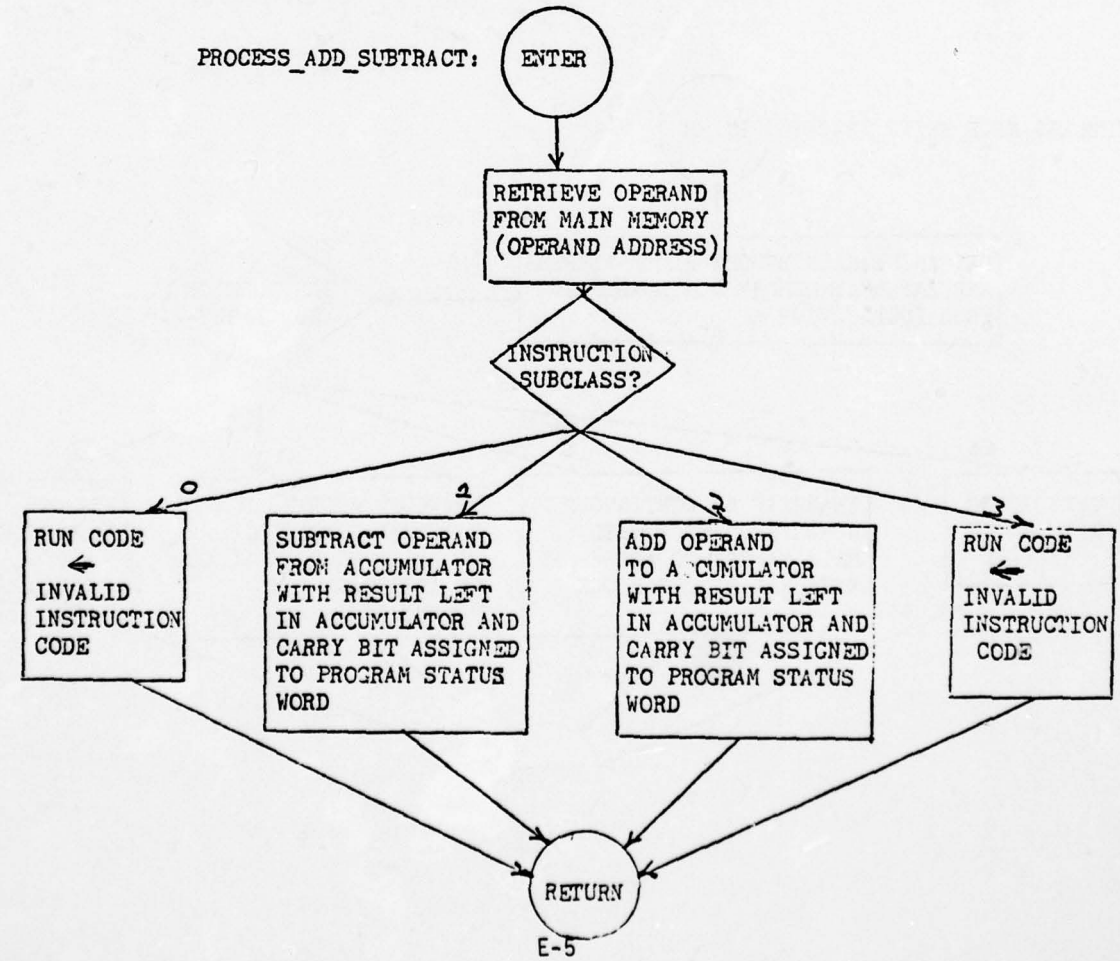
4) Memory access (MAC) is not shown explicitly as a module of the design. Some form of simulated memory access is certainly implied by provisions for instruction fetching, retrieving an operand for adding or subtracting and for loading or storing the accumulator. Also instruction tracing is not shown as an explicit module. It may be highly desirable to implement these features, and others, as explicit subprograms. They are not shown explicitly in the sample design given because it was not deemed necessary to show any logical design for them. (Perhaps such designs would be included in the next step for "top down" design advocates.)

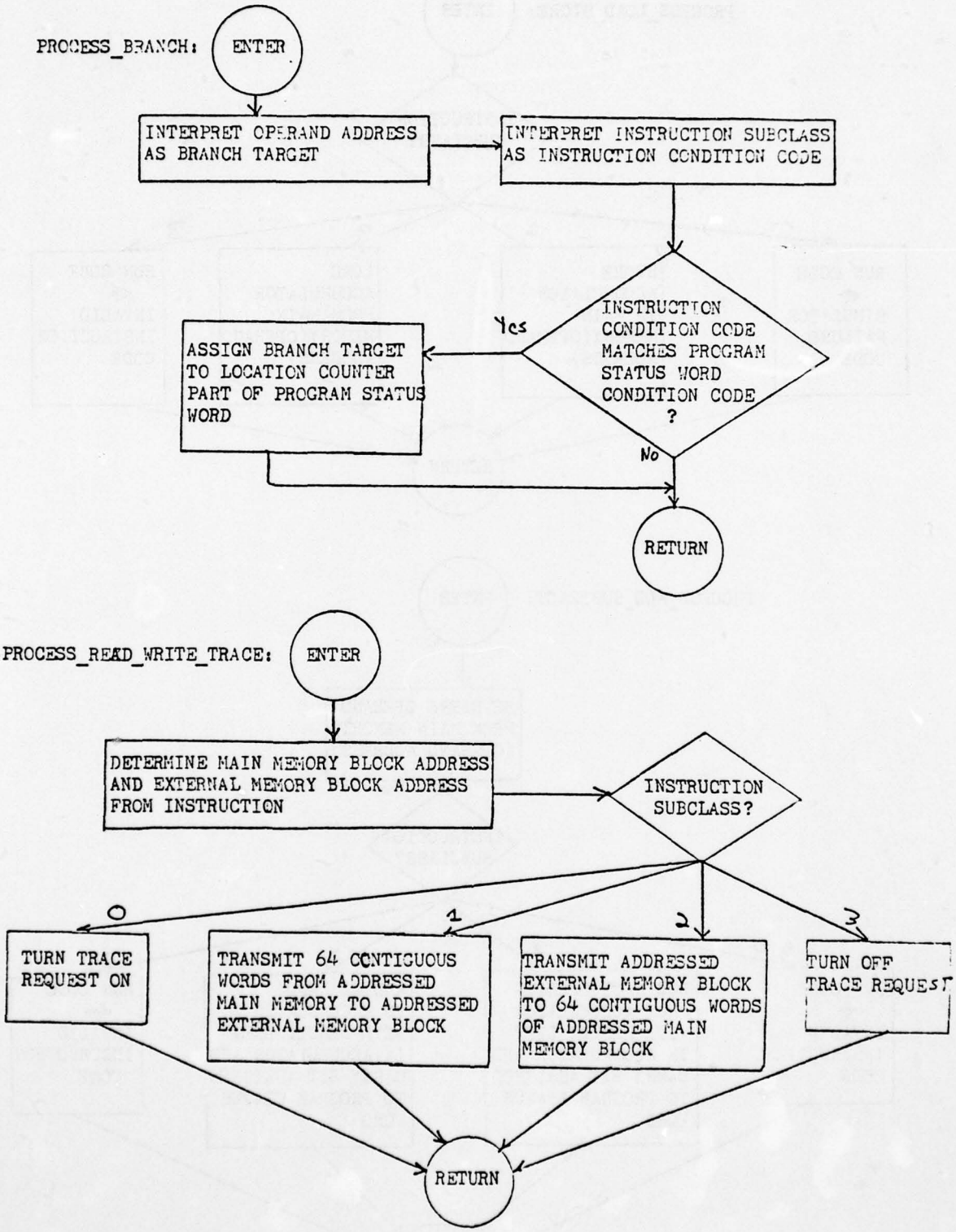
```
SMALL_COMPUTER_SIMULATOR: PROCEDURE;
  INITIALIZE   SIMULATOR;
  DO WHILE   RUN CODE = CONTINUE EXECUTION CODE;
     FETCH INSTRUCTION;
     INCREMENT LOCATION COUNTER PART OF PROGRAM STATUS WORD;
     IF INSTRUCTION IS "HALT" INSTRUCTION THEN RUN CODE ← HALT TERMINATION CODE;
     ELSE
        DO;
           DETERMINE INSTRUCTION CLASS, INSTRUCTION SUBCLASS AND
           OPERAND ADDRESS FROM INSTRUCTION;
           DO CASE( INSTRUCTION CLASS );
              CASE0:
                 CALL PROCESS_LOAD_STORE;
              CASE1:
                 CALL PROCESS_ADD_SUBTRACT;
              CASE2:
                 CALL PROCESS_BRANCH;
              CASE3:
                 CALL PROCESS_READ_WRITE_TRACE;
           END CASE;
        END;
     IF TRACE IS REQUESTED THEN PERFORM TRACE;
     INCREMENT INSTRUCTION EXECUTION COUNT;
     IF INSTRUCTION EXECUTION COUNT EXCEEDS EXECUTION LIMIT THEN
        RUN CODE ← EXECUTION LIMIT TERMINATION CODE;
  END;
  PRINT SIMULATOR STATUS INFORMATION UPON TERMINATION;
END SMALL_COMPUTER_SIMULATOR;



PROCESS_LOAD_STORE: PROCEDURE;
     DO CASE( INSTRUCTION SUBCLASS );
        CASE0:
           RUN CODE ← SIMULATOR FAILURE CODE;
        CASE1:
           STORE ACCUMULATOR IN MAIN MEMORY(OPERAND ADDRESS);
        CASE2:
           LOAD ACCUMULATOR FROM MAIN MEMORY(OPERAND ADDRESS);
        CASE3:
           RUN CODE ← INVALID INSTRUCTION CODE;
     END CASE;
     RETURN TO CALLER;
END PROCESS_LOAD_STORE;
```

```
PROCESS_ADD_SUBTRACT: PROCEDURE;
     RETRIEVE OPERAND FROM MAIN MEMORY(OPERAND ADDRESS);
     DO CASE( INSTRUCTION SUBCLASS );
        CASE0:
           RUN CODE ← INVALID INSTRUCTION CODE;
        CASE1:
           SUBTRACT OPERAND FROM ACCUMULATOR WITH RESULT LEFT
           IN ACCUMULATOR AND CARRY BIT ASSIGNED TO
           PROGRAM STATUS WORD;
        CASE2:
           ADD OPERAND TO ACCUMULATOR WITH RESULT LEFT IN
           ACCUMULATOR AND CARRY BIT ASSIGNED TO PROGRAM
           STATUS WORD;
        CASE3:
           RUN CODE ← INVALID INSTRUCTION CODE;
     END CASE;
     RETURN TO CAILER;
RND PROCESS_ADD_SUBTRACT;

PROCESS_BRANCH: PROCEDURE;
     INTERPRET OPERAND ADDRESS AS BRANCH TARGET;
     INTERPRET INSTRUCTION SUBCLASS AS INSTRUCTION CONDITION CODE;
     IF INSTRUCTION CONDITION CODE MATCHES PROGRAM STATUS WORD
        CONDITION CODE THEN ASSIGN BRANCH TARGET TO LOCATION
        COUNTER PART OF PROGRAM STATUS WORD;
     RETURN TO CALLER;
END PROCESS_BRANCH;

PROCESS_READ_WRITE_TRACE;
     DETERMINE MAIN MEMORY BLOCK ADDRESS AND EXTERNAL MEMORY BLOCK
     ADDRESS FROM INSTRUCTION;
     DO CASE( INSTRUCTION SUBCLASS );
        CASE0:
           TURN TRACE REQUEST ON;
        CASE1:
           TRANSMIT 64 CONTIGUOUS WORDS FROM ADDRESSED MAIN MEMORY
           BLOCK TO ADDRESSED EXTERNAL MEMORY BLOCK;
        CASE2:
           TRANSMIT ADDRESSED EXTERNAL MEMORY BLOCK TO 64 CONTIGUOUS
           WORDS OF ADDRESSED MAIN MEMORY BLOCK;
        CASE3:
           TURN OFF TRACE REQUEST;
     END CASE
     RETURN TO CALLER;
END PROCESS_READ_WRITE_TRACE;
```


SMALL_COMPUTER_SIMULATOR:
START
INITIALIZE SIMULATOR
EXIT
A
RUN CODE = CONTINUE EXECUTION CODE?
No
PRINT SIMULATOR STATUS INFORMATION
Yes
FETCH INSTRUCTION
INCREMENT LOCATION COUNTER PART OF PROGRAM STATUS WORD
RUN CODE ← HALT TERMINATION CODE
Yes
"HALT" INSTRUCTION?
No
INSTRUCTION CLASS
0
1
2
3
PROCESS_LOAD_STORE
PROCESS_ADD_SUBTRACT
PROCESS_BRANCH
PROCESS_READ_WRITE_TRACE
PERFORM TRACE
Yes
TRACE REQUESTED?
B
No
INCREMENT INSTRUCTION EXECUTION COUNT
INSTRUCTION EXECUTION COUNT EXCEEDS EXECUTION LIMIT?
Yes
RUN CODE ← EXECUTION LIMIT TERMINATION CODE
No
A


PROCESS_LOAD_STORE:
ENTER
INSTRUCTION SUBCLASS?
0
1
2
3
RUN CODE ⇐ SIMULATOR FAILURE CODE
STORE ACCUMULATOR IN MAIN MEMORY(OPERAND ADDRESS)
LOAD ACCUMULATOR FROM MAIN MEMORY(OPERAND ADDRESS)
RUN CODE ⇐ INVALID INSTRUCTION CODE
RETURN


PROCESS_ADD_SUBTRACT:
ENTER
RETRIEVE OPERAND FROM MAIN MEMORY (OPERAND ADDRESS)
INSTRUCTION SUBCLASS?
0
1
2
3
RUN CODE ⇐ INVALID INSTRUCTION CODE
SUBTRACT OPERAND FROM ACCUMULATOR WITH RESULT LEFT IN ACCUMULATOR AND CARRY BIT ASSIGNED TO PROGRAM STATUS WORD
ADD OPERAND TO ACCUMULATOR WITH RESULT LEFT IN ACCUMULATOR AND CARRY BIT ASSIGNED TO PROGRAM STATUS WORD
RUN CODE ⇐ INVALID INSTRUCTION CODE
RETURN


PROCESS_BRANCH:
ENTER
INTERPRET OPERAND ADDRESS AS BRANCH TARGET
INTERPRET INSTRUCTION SUBCLASS AS INSTRUCTION CONDITION CODE
INSTRUCTION CONDITION CODE MATCHES PROGRAM STATUS WORD CONDITION CODE ?
Yes
ASSIGN BRANCH TARGET TO LOCATION COUNTER PART OF PROGRAM STATUS WORD
No
RETURN
PROCESS_READ_WRITE_TRACE:
ENTER
DETERMINE MAIN MEMORY BLOCK ADDRESS AND EXTERNAL MEMORY BLOCK ADDRESS FROM INSTRUCTION
INSTRUCTION SUBCLASS?
0
1
2
3
TURN TRACE REQUEST ON
TRANSMIT 64 CONTIGUOUS WORDS FROM ADDRESSED MAIN MEMORY TO ADDRESSED EXTERNAL MEMORY BLOCK
TRANSMIT ADDRESSED EXTERNAL MEMORY BLOCK TO 64 CONTIGUOUS WORDS OF ADDRESSED MAIN MEMORY BLOCK
TURN OFF TRACE REQUEST
RETURN

APPENDIX F - BASIC DESCRIPTION OF DESIGN PROBLEM

Assembler Language Translator Design Problem

An assembler language for the hypothetical "small computer" (Small Computer Assembler Language - SCAL) is outlined below. You will be given a software design assignment requirement for a two pass translator for SCAL. One pass of the translator is to be designed with one of the two software design methods presented in class (flowcharts and PDL's). The other pass is to be designed using the other method.

Half the class will be required to use flowcharts for the first pass and half the class will be required to use PDL. The designs from those students consenting to participate in the software design study being conducted this semester will be used (anonymously) for part of this study.

Written requirements for each pass are given on a separate handout. The pass one assignment will be distributed in class Feb. 10 and will be due Feb. 15. (Turn it in to Dr. Fisher's secretary.) The pass two assignment is to be picked up in Dr. Fisher's office when you turn in the pass one part. The pass two design will be due Feb. 17.

SCAL features:

1) Symbolic machine instructions with instruction formats

Optional label	HALT	Optional symbolic or absolute address
" "	LOAD	Symbolic or absolute address
" "	STORE	" " " "
" "	ADD	" " " "
" "	SUB	" " " "
" "	B	Numeric condition code(0-3), symbolic or absolute address
" "	READ	Symbolic or absolute address
" "	WRITE	" "

2) Constant definitions with formats

Optional label	DC	Integer constant
" "	DIO	Symbolic or absolute memory block address, symbolic or absolute external store address

Note: DIO is intended for defining the addresses for READ or WRITE operations. Ordinarily a DIO would be referenced only by a READ or WRITE. Ordinarily a symbolic reference in a READ or WRITE would be a reference to a DIO. Neither of these are required, however.

3) Assembler instructions with formats

	PROG	Absolute address (starting location of program)
Required label	EQUATE	Absolute address (address value of label symbol)
	LOC	Absolute address (reassigns assembler location counter for location of following instructions)
Optional label	DS	Positive integer value (number of words of storage reserved)
	END	Symbolic or absolute address (address of first instruction to be executed)

The symbolic addresses are very simple in form and do not allow address expressions such as *+3 or SUM+2. Neither are literal operands allowed. All absolute addresses or numeric constants are assumed to be in decimal form.

Comments and observations about the assignment:

You are encouraged to design your logic at a level similar to that displayed in the software design illustrations. For example, don't be concerned with the detailed logic for " recognizing" a valid symbol or constant (lexical analysis or scanning). You should be able to complete your design in 5 $8\frac{1}{2}$ x 11 pages or less.

Assume that symbol table design and processing algorithms already exist; that is to say, do not attempt to design them. Further assume that the symbol table structure is very simple in that no "type" codes are required - just symbols and absolute addresses.

Neither the PDL nor the flowchart design method provides for data structure definition. For the most part data structure requirements for the assembler are elementary and may be implied by appropriate wordage and symbolism. If certain tables or data structures used in your design are sufficiently complex that separate descriptions of them are needed, then provide them.

EXAMPLE PASS 1

EXAMPLE PASS 1 (cont'd)

HEX LOC	IMAGE INSTR	ERR & CNTL	INPUT LOC	OP	ADDR
				PRØG	100
64	E 000		LB37	READ	IAD1
65	E 000			READ	IAD2
66	2 000			LØAD	TSTI
67	1 000			STØRE	TEST
68	2 000		LB1	LØAD	LB10
69	1 000			STØRE	LB2
6A	2 000			LØAD	LB11
6B	1 000			STØRE	LB4
6C	2 000			LØAD	ZERØ
6D	1 000			STØRE	C
6E	2 000		LB2	LØAD	A
6F	5 000		LB4	SUB	B
70	6 000			ADD	C
71	1 000			STØRE	C
72	2 000			LØAD	LB2
73	6 000			ADD	ØNE
74	1 000			STØRE	LB2
75	2 000			LØAD	LB4
76	6 000			ADD	ØNE
77	1 000			STØRE	LB4
78	2 000			LØAD	TEST
79	6 000			ADD	ØNE
7A	1 000			STØRE	TEST
7B	8 000			BR 0	LB3
7C	2 000			LØAD	MAX

HEX LOC	IMAGE INSTR	ERR & CNTL	INPUT LOC	OP	ADDR	
7D	6 000			ADD	ØNE	
7E	8 000			BR 0	LB2	
7F	D 000		LB3	WRITE	ØAD1	
80	0 000			HALT		
				LØC	200	
C8	0 000		~~ZERØ~~	DC	0	
C9	0 001		ØNE	DC	1	
CA	0 000		IAD1	DIØ	10,E1	
CB	0 000		IAD2	DIØ	M2,99	
CC	0 000		ØAD1	DIØ	C1,10	
CD	2 000		LB10	LØAD	A	
CE	5 000		LB11	SUB	B	
CF	0 FC0		TSTI	DC	4032	
D0	0 FFF		MAX	DC	4095	
			A	EQUATE	640	Error (Listing problem also)
280	0 000		A	DS	64	
2C0	0 080		TEST	DC	0	
			E1	EQUATE	89	
			M2	EQUATE	29	
				LØC	576	
240	0 000		B	DS	64	
			C1	EQUATE	4	
280			C	DC	0	
				END	LB37	

HEX Loc	IMAGE INSTR	ERR & CNTL	INPUT LOC	IMAGE OP	ADDR
				PROG	100
64	E 0CA		LB37	READ	IADI
65	E 0CB			READ	IAD1
66	2 0CF			LOAD	TSTI
67	1 2C0			STORE	TEST
68	2 0CD		LB1	LOAD	LB10
69	1 06E			STORE	LB2
6A	2 0CE			LOAD	LB11
6B	1 06F			STORE	LB4
6C	2 068			LOAD	ZERO
6D	1 100			STORE	C
6E	2 280		LB2	LOAD	A
6F	5 240		LB4	SUB	B
70	6 100			ADD	C
71	1 100			STORE	C
72	2 06E			LOAD	LB2
73	6 0C9			ADD	ONE
74	1 06E			STORE	LB2
75	2 06F			LOAD	LB4
76	6 069			ADD	ONE
77	1 06F			STORE	LB4
78	2 2C0			LOAD	TEST
79	6 0C9			ADD	ONE
7A	1 2C0			STORE	TEST
7B	8 07F			BRO	LB3
7C	2 0C0			LOAD	MAX

HEX Loc	IMAGE DISTR	ERR & CNTL	INPUT LOC	IMAGE OP	ADDR
7D	6 064			ADD	ONE
7E	8 065			BRO	LB2
7F	D 000		LB3	WRITE	OADI
80	0 000			HALT	
				LOC	200
C8	0 000		ZERO	DC	0
C9	0 001		ONE	DC	1
CA	2 859		IAD1	DIO	10, E1
CB	7 463		IAD2	DIO	M2,99
CC	1 00A		OADI	DIO	C1, 10
CD	2 220		LB10	LOAD	A
CE	5 240		LB11	SUB	B
CF	0 FC0		TSTI	DC	4032
D0	0 FFF		MAX	DC	4095
			A	EQUATE	100
280	0 000		A	DS	64
2C0	0 000		TEST	DC	0
			E1	EQUATE	59
			M2	EQUATE	29
				LOC	5F6
240	0 000		B	DS	64
			C1	EQUATE	4
280			C	DC	0
				END	LB37

UNSØRTED

064	LB37	230	A---
068	LB1-	240	B---
06E	LB2-	100	C---
06F	LB4-	004	C1--
07F	LB3-	002	B1--
0C8	ZERØ	0CA	IAD1
0C9	ØNE-	0C8	IAD2
0CA	IAD1	068	LB1-
0CB	IAD2	0CD	LB10
0CC	ØAD1	0CE	LB11
0CD	LB10	06E	LB2-
0CE	LB11	07F	LB3-
0CF	TSTI	064	LB37
0D0	MAX-	06F	LB4-
280	A---	0D0	MAX-
2C0	TEST	01D	M2--
059	E1--	0CC	ØAD1
01D	M2--	0C9	ØNE-
240	B---	2C0	TEST
100	C---	0CF	TSTI
004	C1--	0C8	ZERO

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX G - DESCRIPTION OF PASS 1 DESIGN PROBLEM

Assembler language Translator Design Problem, Pass 1

Name: _____ Design Method: Flowchart / PDL

Pass one has the following responsibilities:

To read, process and copy the SCAL program to a secondary storage file in the form suggested in the example handout.

To fill in the symbol table for defined symbols.

To detect invalid operation codes. (Substitute a "HALT 0" image for these.)

To detect multiple definitions of a symbolic address.

To detect "out of range address" situatuons.

To detect other errors and take error action consistent with your overall software design.

Note 1: The copy of the SCAL program will serve as input to the second pass.

Note 2: Assume that all necessary information is passed in a global sense to pass two; that is, it isn't necessary to identify explicit parameters.

Note 3: Make sure you fill out and turn in the time questionaire if you are participating in the software design experiment. This information is needed for the study.

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX H - DESCRIPTION OF PASS 2 DESIGN PROBLEM

Assembler Language Translator Design Problem, Pass 2

Name: _____ Design Method: Flowchart / PDL

Pass two has the follwoing responsibilities:

To generate absolute object code in the format indicated below on a secondary file.

To create a printed form of the orginal program with machine code information and error codes as suggested by the example handout.

To detect undefined symbolic addresses. (Substitute a zero for such an address in the generated code.)

To detect "out of range" absolute addresses and constants.

To detect errors that shouldn't occur if the first pass is working correctly.

To detect other errors and take error action consistent with your overall software design.

Format of generated code - 40 word records

1st word	-	absolute addres where machine language text is to be loaded
2nd word	-	number of words of machine language text following
Words 3 to 40	-	machine language text

"trailer" record

1st word	-	"0"
2nd word	-	"0"
3rd word	-	Absolute address of starting instruction

Note: Make sure you fill out and turn in the time questionaire. This information is needed for the software design study.

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX I - TIME EXPENDITURE QUESTIONNAIRE, DESIGN PHASE

TIME QUESTIONAIRE
ASSEMBLER DESIGN PROBLEM, PASS 1

Name:_____

Date:_____ Code:_____
(Leave blank)

Please record the time you have spent on your class project in the last week, excluding time spent in class. Record time in hours and/or minutes by category.

hrs : min

Time spent in:

Preparing initial design . . . . . ___:___

Revising design . . . . . ___:___

Hand-checking design . . . . . ___:___

Preparing documentation . . . . . ___:___

Just thinking about project . . . . . ___:___

Other (indicate nature of activity)

_____ ___:___

_____ ___:___

This information is solely for experimental purposes; your responses will in no way affect your course grade. Please be as accurate as possible.

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX J - PDL-FLOWCHART QUESTIONNAIRE

Name:_____

INSTRUCTIONS

This questionnaire is designed solely for experimental purposes; your responses will in no way affect your course grade. The goal of this questionnaire is to provide us with an objective, unbiased comparison of flowcharts and program design languages (PDLs) as methods for the development and documentation of software designs. Please complete these questions honestly and thoughtfully, so that your answers reflect your opinions about these methods based on the experience you have had with them.

Most of the questions which follow ask you to compare the two methods by checking a box on a scale. In every case, check one and only one box, and place your checks within the box, rather than on boundaries between boxes.

Code: _____

1. From the viewpoint of the software designer who must use flowcharts or a program design language to document his design, how do the two compare?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

2. From the viewpoint of the programmer who must understand and implement a design expressed in either a program design language or flowcharts, how do the two compare?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

3. How do the methods compare with respect to helping the software designer understand his problem?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

4. With respect to encouraging the use of "good" design practices?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

5. Ease of preparation of design documentation?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

6. Ease of modification of design documentation?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

7. Ease of modification of design?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

8. Ease of expression of control flow by designer?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

9. Ease of expression of data flow?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

10. From the viewpoint of the programmer who must understand and implement the design, how do the methods compare on comprehensibility?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

11. How do they compare on ease of reading (i.e., effort involved, speed, etc.)?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

12. Ease of understanding control flow?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

13. Ease of understanding data flow?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

14. Ease of implementing in a programming language?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

15. As the designer develops his design, he may notice that it contains errors or undesirable properties. How do the documentation methods compare with respect to helping the designer detect such problems in his design?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

16. How do the methods compare with respect to helping the designer notice that the same operation is being performed redundantly at several points in the system he is designing?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

17. With respect to helping him/her notice that a module is too big, and should be broken into several modules?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

18. Helping him/her notice that a module is logically too complex?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

19. That a module has too many or too few submodules?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

20. That a module contains several logically unrelated functions (i.e., that the module is not cohesive)?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

21. That a module has not been defined in enough detail?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

22. That a module has been defined in too much detail?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

23. That a module is not functionally complete (i.e., that necessary operations are missing)?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

24. That modules are not sufficiently independent, and are communicating with each other more than they should (i.e., that modules are tightly coupled)?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

25. That the overall design is too complex?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

26. That modules which access the same data are widely separated in the design (data fragmentation)?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

27. That a module is trivial, and should be compressed into another module?

Flowchart absolutely better	Flowchart much better	Flowchart moderately better	Flowchart slightly better	PDL slightly better	PDL moderately better	PDL much better	PDL absolutely better

28. Questions 16 through 27 have asked you how flowcharts and program design languages compare with respect to helping the designer notice a number of specific problems in his/her design. In general, did you find these questions (pick one):

( ) easy to answer.

( ) difficult to answer because you don't see any difference between the documentation methods with respect to these criteria.

( ) Difficult to answer because you couldn't visualize the kinds of design errors and problems referred to in the questions.

29. Overall, how do you think the two documentation methods compare?

PDL absolutely better	PDL much better	PDL moderately better	PDL slightly better	Flowchart slightly better	Flowchart moderately better	Flowchart much better	Flowchart absolutely better

30. Are there any other documentation methods you think are better than either of these?

_____

_____

_____

_____

31. Please write any other comments you have on these two methods (or others you think are superior). In using both methods for a design problem, you may very well have acquired some insights we don't have. Therefore, please comment on even obvious things if they have not been addressed in this questionnaire.

APPENDIX K - SIMULATOR DESIGN, FLOWCHART FORM

Name____ _____

COMSC 5113 SIMULATOR DESIGN Spring 1977

Explanatory notes:

1) The location counter in this design generally points to the next instruction in contrast to the current instruction as stated in the microNova manual. This tends to simplify the design but has implications concerning tracing and dumping.

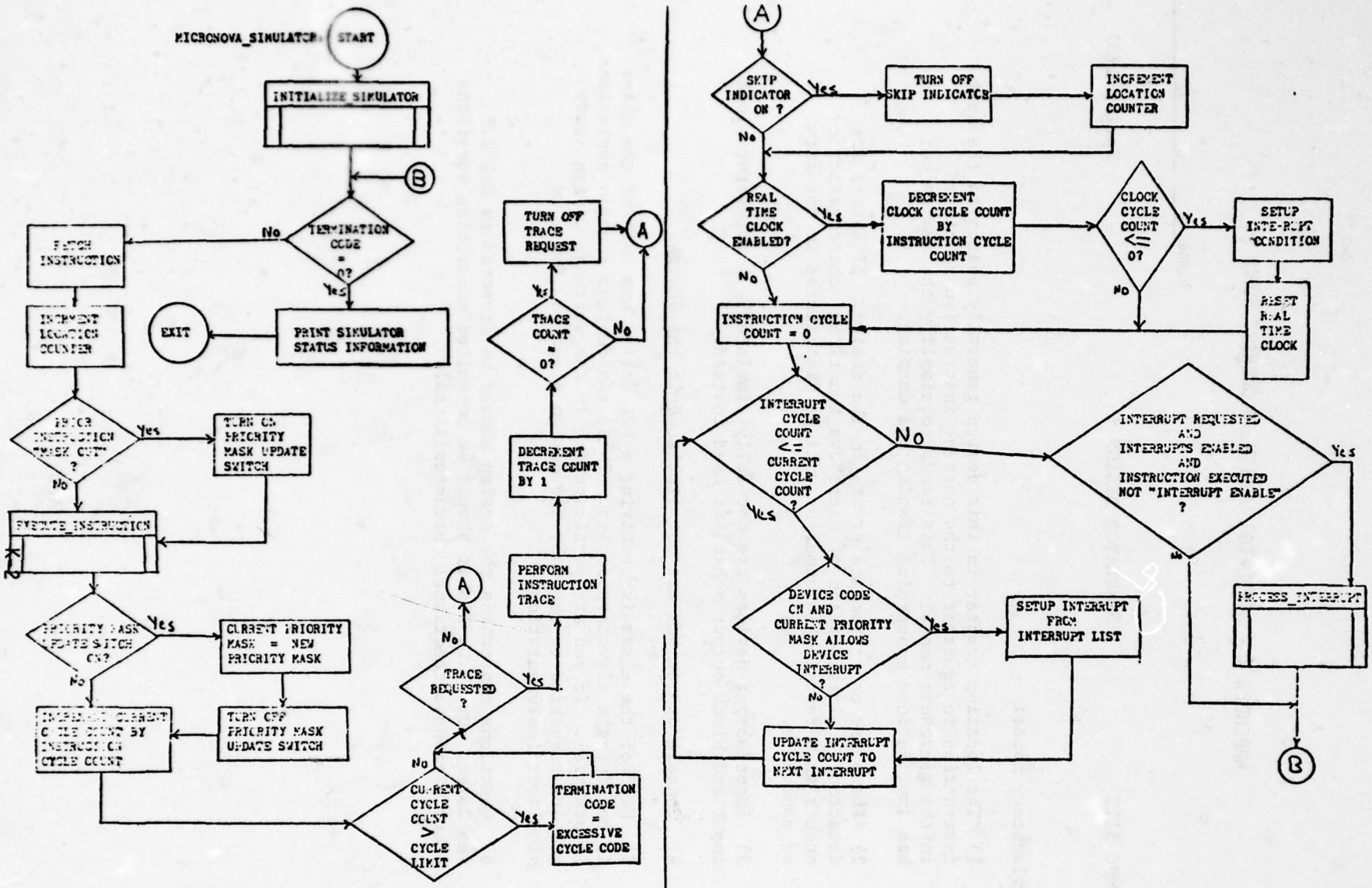
2) There are no deliberate errors in the design. If errors are detected or clarification of microNova functional characteristics occurs then revision information will be distributed in the form of memoranda.

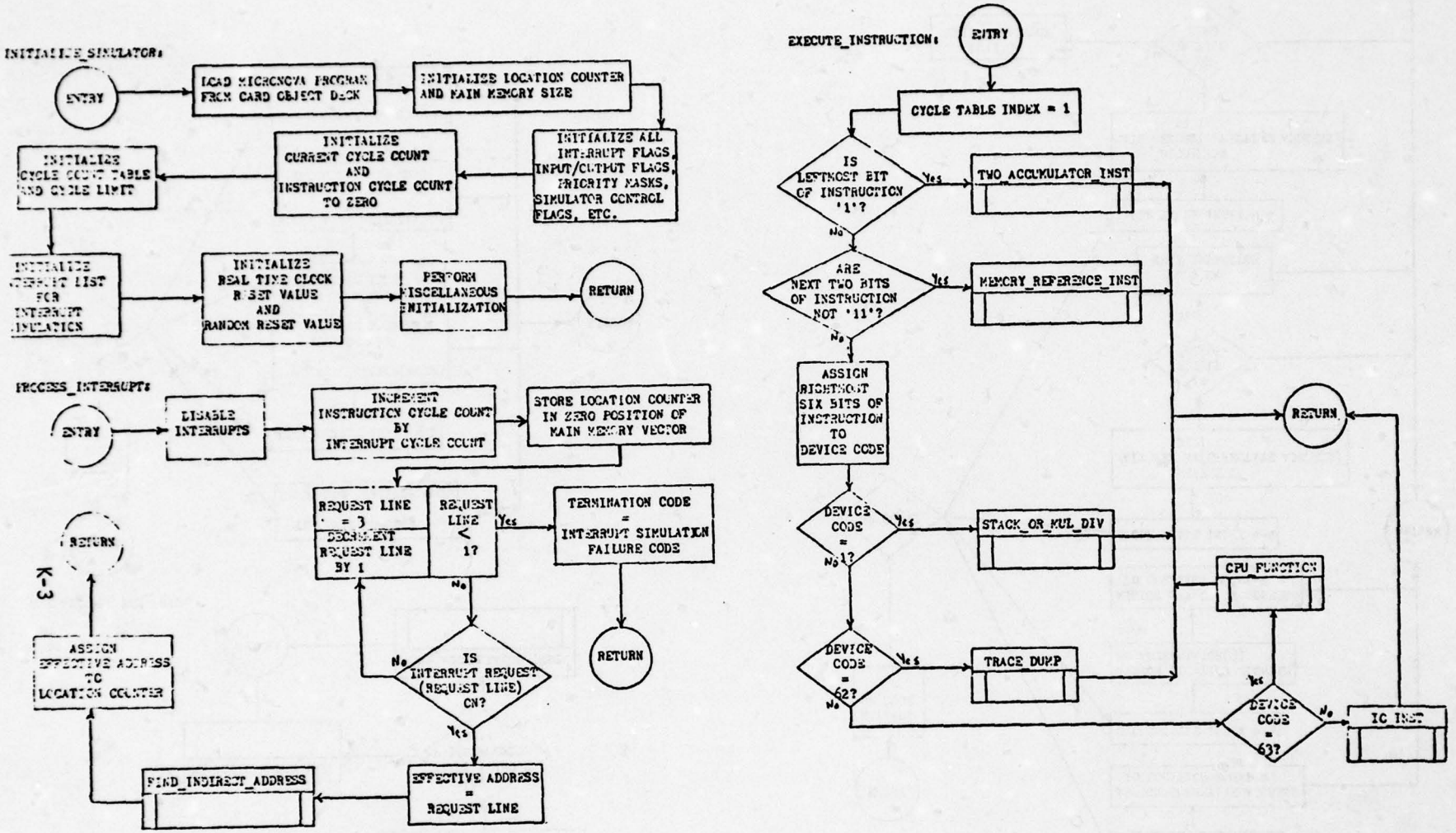
3) Input/output devices are not fully simulated. A simulator block input and block output scheme is used instead.

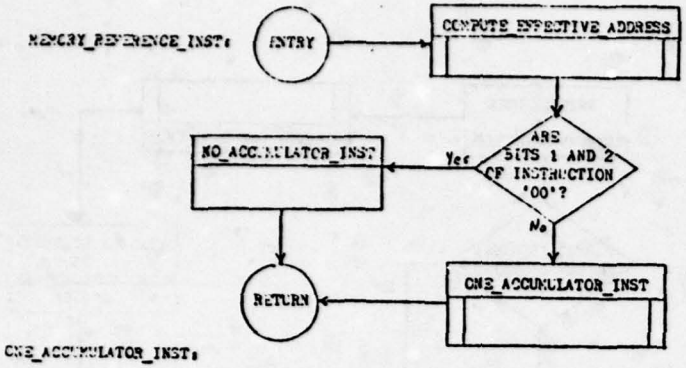
4) Console operations are not simulated in the design.

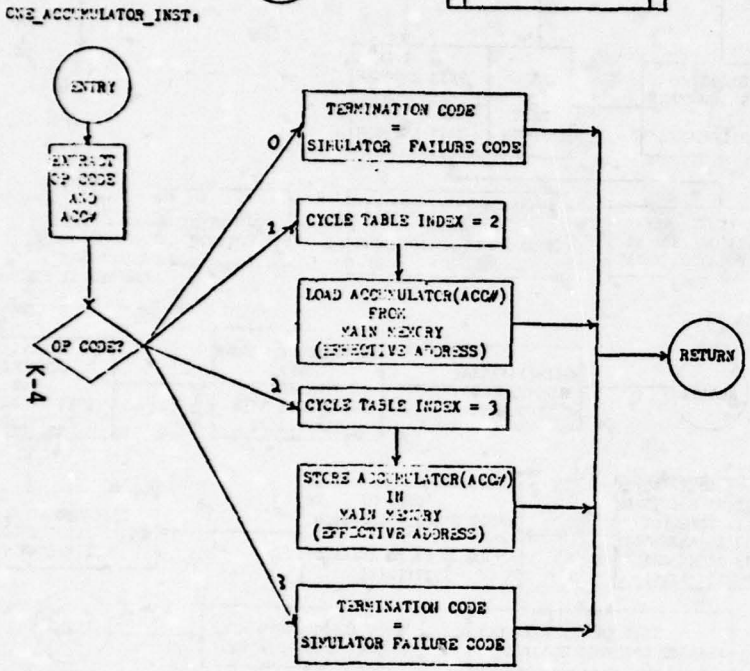
5) Half of the class is receiving a PDL design and half of the class is receiving a flowchart design. These two designs contain equivalent information. If you are participating in the software design study you are requested to use only the design given you for your simulator implementation.

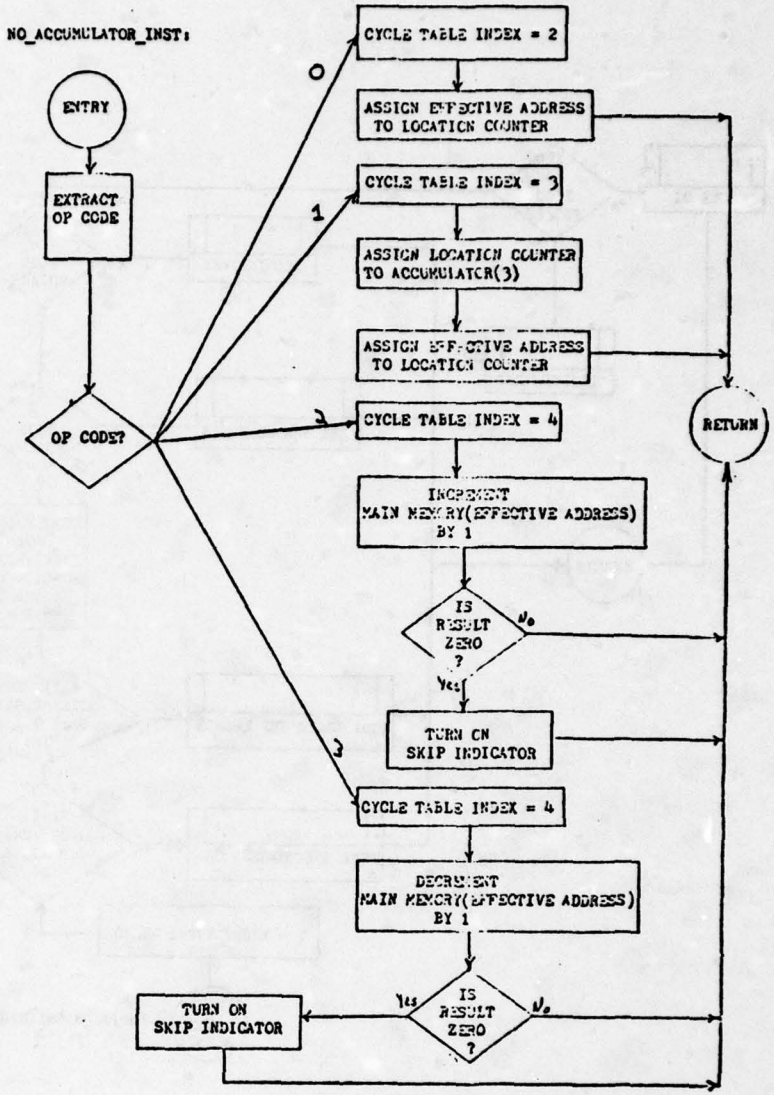
6) Questions concerning the design should be directed to Dr. J.R. Van Doren. Dr. D.D. Fisher should be consulted concerning questions about microNova functional characteristics.

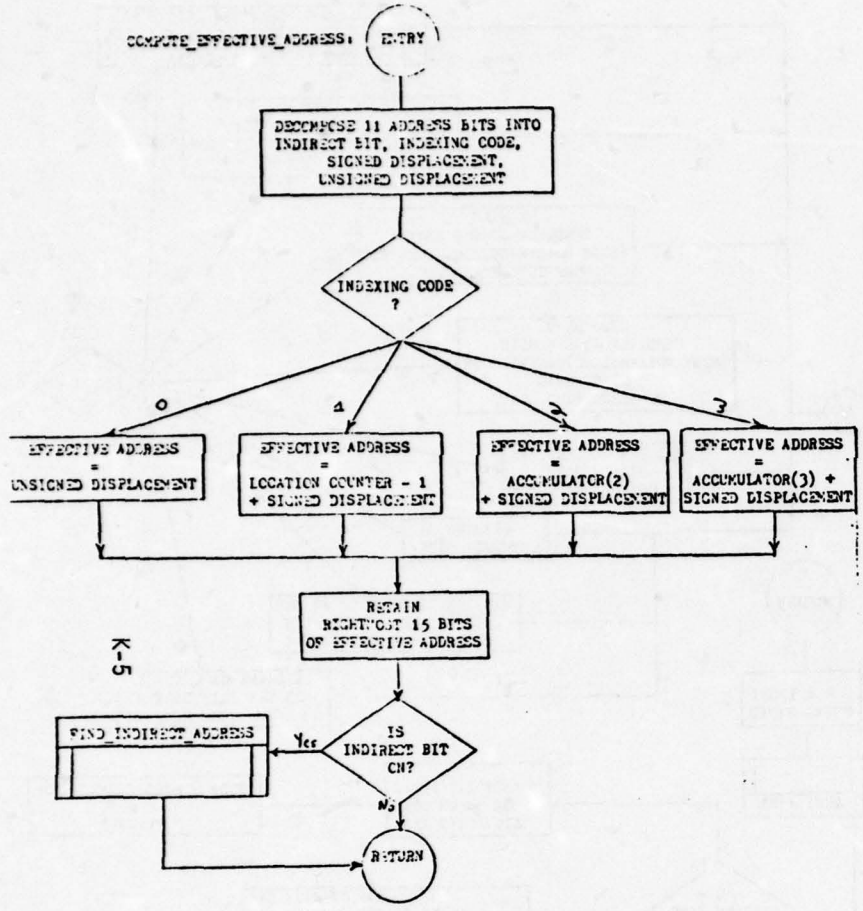

MICRONOVA_SIMULATOR
START
INITIALIZE SIMULATOR
B
TERMINATION CODE = 0?
No
Yes
FETCH INSTRUCTION
INCREMENT LOCATION COUNTER
EXIT
PRINT SIMULATOR STATUS INFORMATION
PRIOR INSTRUCTION MASK OUT ?
Yes
No
TURN ON PRIORITY MASK UPDATE SWITCH
EXECUTE_INSTRUCTION
PRIORITY MASK UPDATE SWITCH ON?
CURRENT PRIORITY MASK = NEW PRIORITY MASK
TURN OFF PRIORITY MASK UPDATE SWITCH
CURRENT CYCLE COUNT > CYCLE LIMIT
TERMINATION CODE = EXCESSIVE CYCLE CODE
TRACE REQUESTED ?
PERFORM INSTRUCTION TRACE
DECREMENT TRACE COUNT BY 1
TRACE COUNT = 0?
TURN OFF TRACE REQUEST
A
SKIP INDICATOR ON ?
TURN OFF SKIP INDICATOR
INCREMENT LOCATION COUNTER
REAL TIME CLOCK ENABLED?
DECREMENT CLOCK CYCLE COUNT BY INSTRUCTION CYCLE COUNT
CLOCK CYCLE COUNT <= 0?
SETUP INTERRUPT CONDITION
RESET REAL TIME CLOCK
INSTRUCTION CYCLE COUNT = 0
INTERRUPT CYCLE COUNT <= CURRENT CYCLE COUNT ?
INTERRUPT REQUESTED AND INTERRUPTS ENABLED AND INSTRUCTION EXECUTED NOT "INTERRUPT ENABLE" ?
PROCESS_INTERRUPT
DEVICE CODE ON AND CURRENT PRIORITY MASK ALLOWS DEVICE INTERRUPT ?
SETUP INTERRUPT FROM INTERRUPT LIST
UPDATE INTERRUPT CYCLE COUNT TO NEXT INTERRUPT

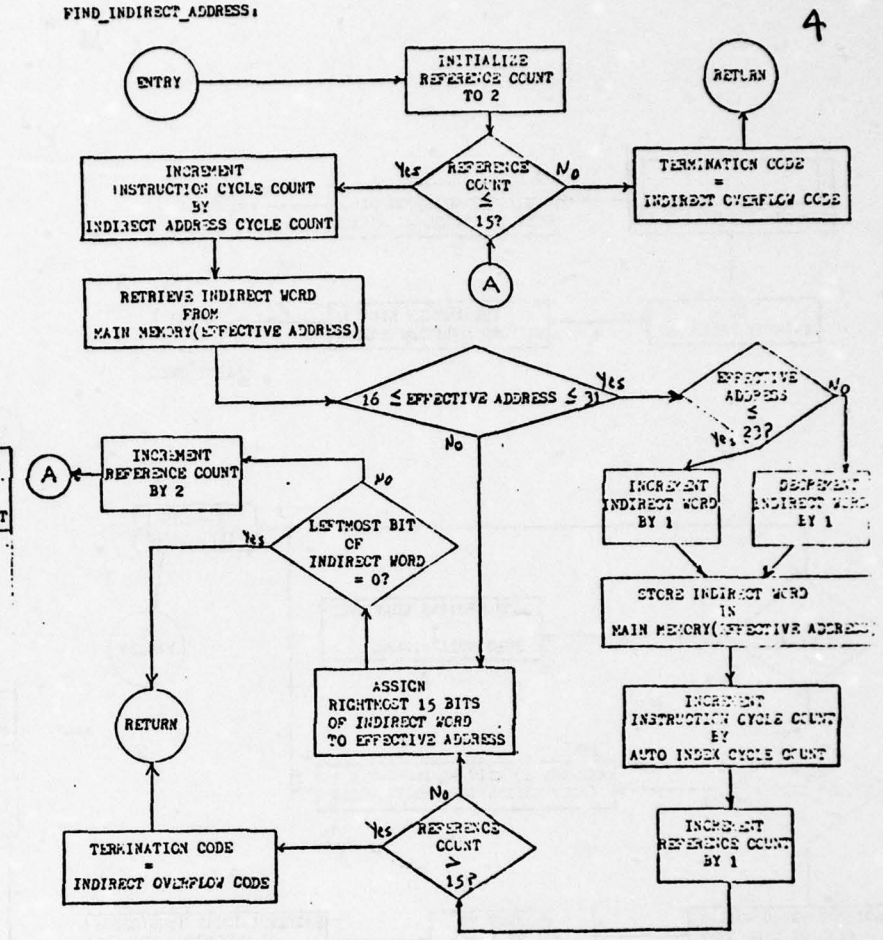

INITIALIZE_SIMULATOR:
ENTRY
LOAD MICRONOVA PROGRAM FROM CARD OBJECT DECK
INITIALIZE LOCATION COUNTER AND MAIN MEMORY SIZE
INITIALIZE ALL INTERRUPT FLAGS, INPUT/OUTPUT FLAGS, PRIORITY MASKS, SIMULATOR CONTROL FLAGS, ETC.
INITIALIZE CURRENT CYCLE COUNT AND INSTRUCTION CYCLE COUNT TO ZERO
INITIALIZE CYCLE COUNT TABLE AND CYCLE LIMIT
INITIALIZE INTERRUPT LIST FOR INTERRUPT SIMULATION
INITIALIZE REAL TIME CLOCK RESET VALUE AND RANDOM RESET VALUE
PERFORM MISCELLANEOUS INITIALIZATION
RETURN
PROCESS_INTERRUPT:
ENTRY
DISABLE INTERRUPTS
INCREMENT INSTRUCTION CYCLE COUNT BY INTERRUPT CYCLE COUNT
STORE LOCATION COUNTER IN ZERO POSITION OF MAIN MEMORY VECTOR
REQUEST LINE = 3
DECREMENT REQUEST LINE BY 1
REQUEST LINE < 17?
Yes
No
TERMINATION CODE = INTERRUPT SIMULATION FAILURE CODE
RETURN
IS INTERRUPT REQUEST (REQUEST LINE) ON?
No
Yes
EFFECTIVE ADDRESS = REQUEST LINE
FIND_INDIRECT_ADDRESS
ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER
RETURN
K13
EXECUTE_INSTRUCTION:
ENTRY
CYCLE TABLE INDEX = 1
IS LEFTMOST BIT OF INSTRUCTION '1'?
Yes
TWO_ACCUMULATOR_INST
No
ARE NEXT TWO BITS OF INSTRUCTION NOT '11'?
Yes
MEMORY_REFERENCE_INST
No
ASSIGN RIGHTMOST SIX BITS OF INSTRUCTION TO DEVICE CODE
DEVICE CODE = 1?
Yes
STACK_OR_MUL_DIV
No
DEVICE CODE = 62?
Yes
TRACE_DUMP
No
DEVICE CODE = 63?
Yes
No
IO_INST
CPU_FUNCTION
RETURN

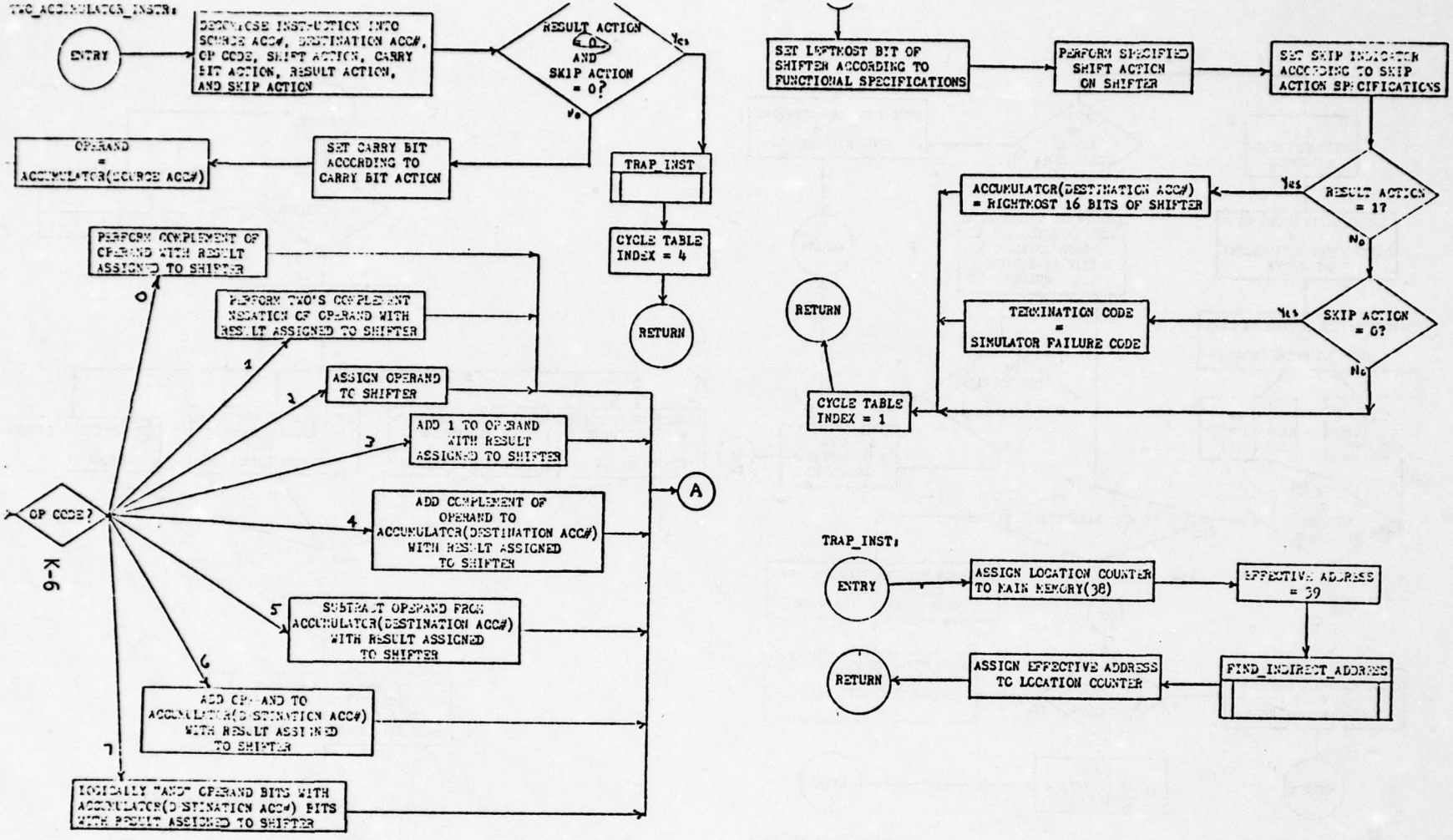

MEMORY_REFERENCE_INST:
ENTRY
COMPUTE EFFECTIVE ADDRESS
ARE BITS 1 AND 2 OF INSTRUCTION '00'?
Yes
No
NO_ACCUMULATOR_INST
ONE_ACCUMULATOR_INST
RETURN

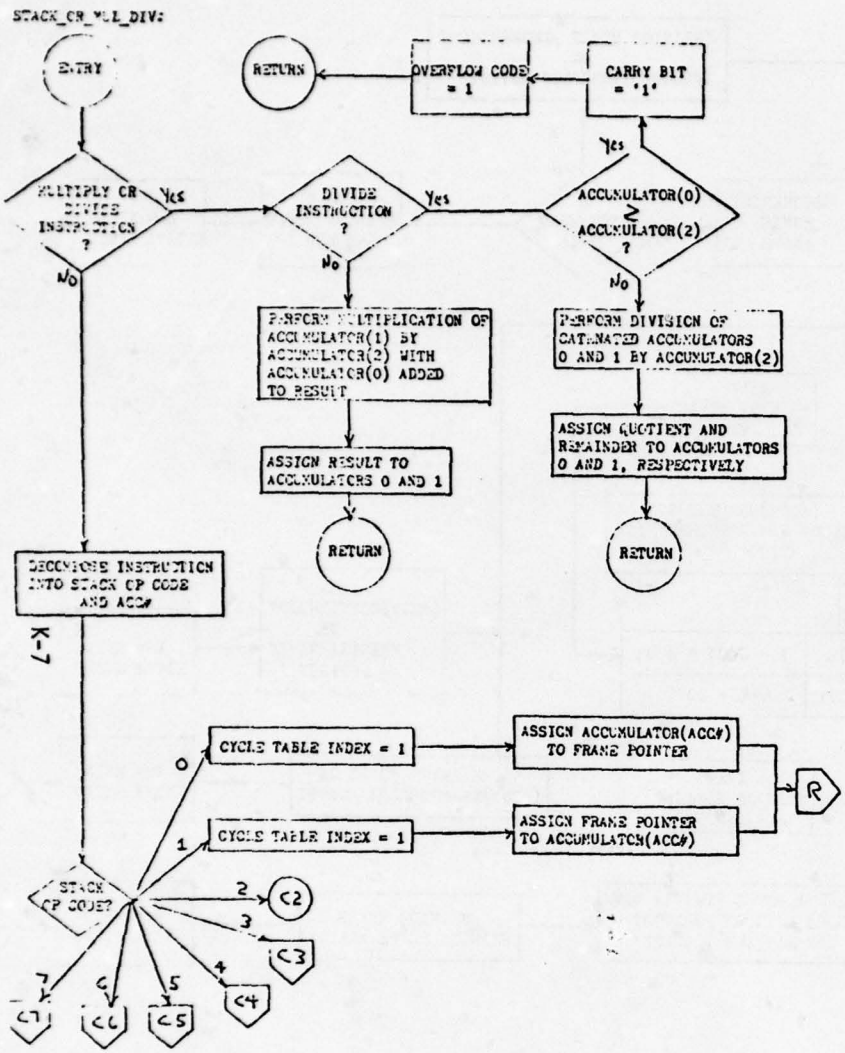

ONE_ACCUMULATOR_INST:
ENTRY
EXTRACT OP CODE AND ACC#
OP CODE?
0
1
2
3
TERMINATION CODE = SIMULATOR FAILURE CODE
CYCLE TABLE INDEX = 2
LOAD ACCUMULATOR(ACC#) FROM MAIN MEMORY (EFFECTIVE ADDRESS)
CYCLE TABLE INDEX = 3
STORE ACCUMULATOR(ACC#) IN MAIN MEMORY (EFFECTIVE ADDRESS)
TERMINATION CODE = SIMULATOR FAILURE CODE
RETURN

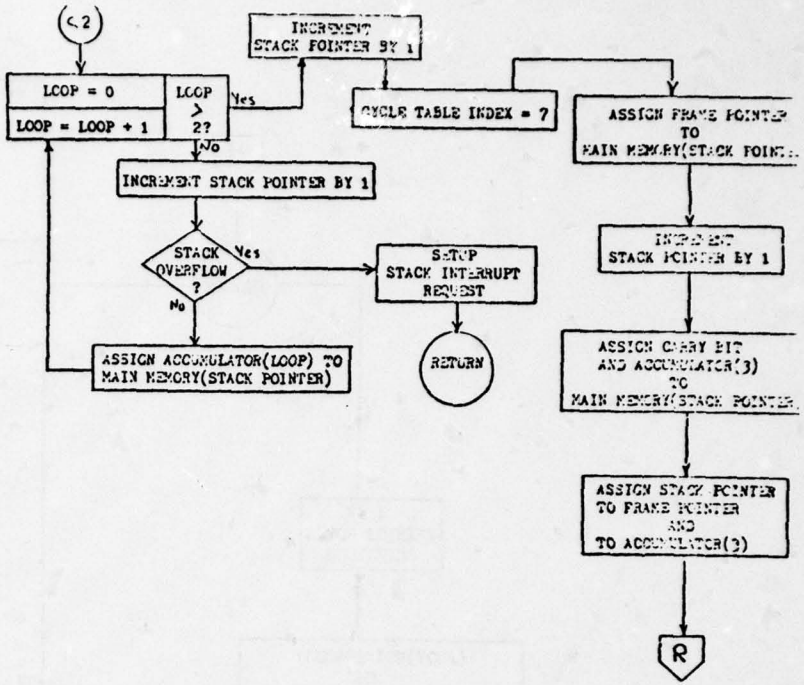

NO_ACCUMULATOR_INST:
ENTRY
EXTRACT OP CODE
OP CODE?
0
1
2
3
CYCLE TABLE INDEX = 2
ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER
CYCLE TABLE INDEX = 3
ASSIGN LOCATION COUNTER TO ACCUMULATOR(3)
ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER
CYCLE TABLE INDEX = 4
INCREMENT MAIN MEMORY(EFFECTIVE ADDRESS) BY 1
IS RESULT ZERO ?
Yes
No
TURN ON SKIP INDICATOR
CYCLE TABLE INDEX = 4
DECREMENT MAIN MEMORY(EFFECTIVE ADDRESS) BY 1
IS RESULT ZERO ?
Yes
No
TURN ON SKIP INDICATOR
RETURN

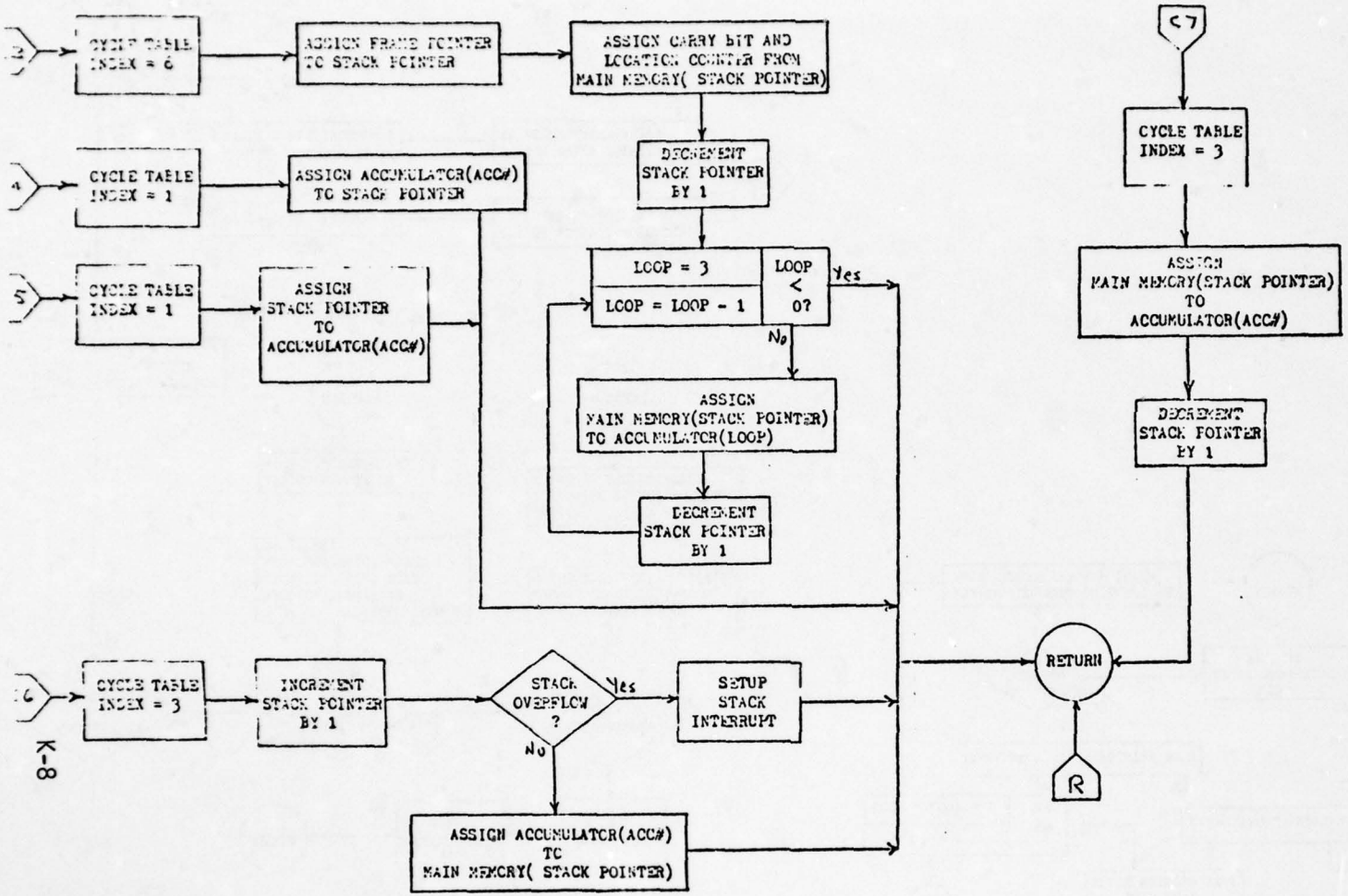

COMPUTE_EFFECTIVE_ADDRESS:
ENTRY
DECOMPOSE 11 ADDRESS BITS INTO INDIRECT BIT, INDEXING CODE, SIGNED DISPLACEMENT, UNSIGNED DISPLACEMENT
INDEXING CODE ?
0
1
2
3
EFFECTIVE ADDRESS = UNSIGNED DISPLACEMENT
EFFECTIVE ADDRESS = LOCATION COUNTER - 1 + SIGNED DISPLACEMENT
EFFECTIVE ADDRESS = ACCUMULATOR(2) + SIGNED DISPLACEMENT
EFFECTIVE ADDRESS = ACCUMULATOR(3) + SIGNED DISPLACEMENT
RETAIN RIGHTMOST 15 BITS OF EFFECTIVE ADDRESS
IS INDIRECT BIT ON?
Yes
No
FIND_INDIRECT_ADDRESS
RETURN

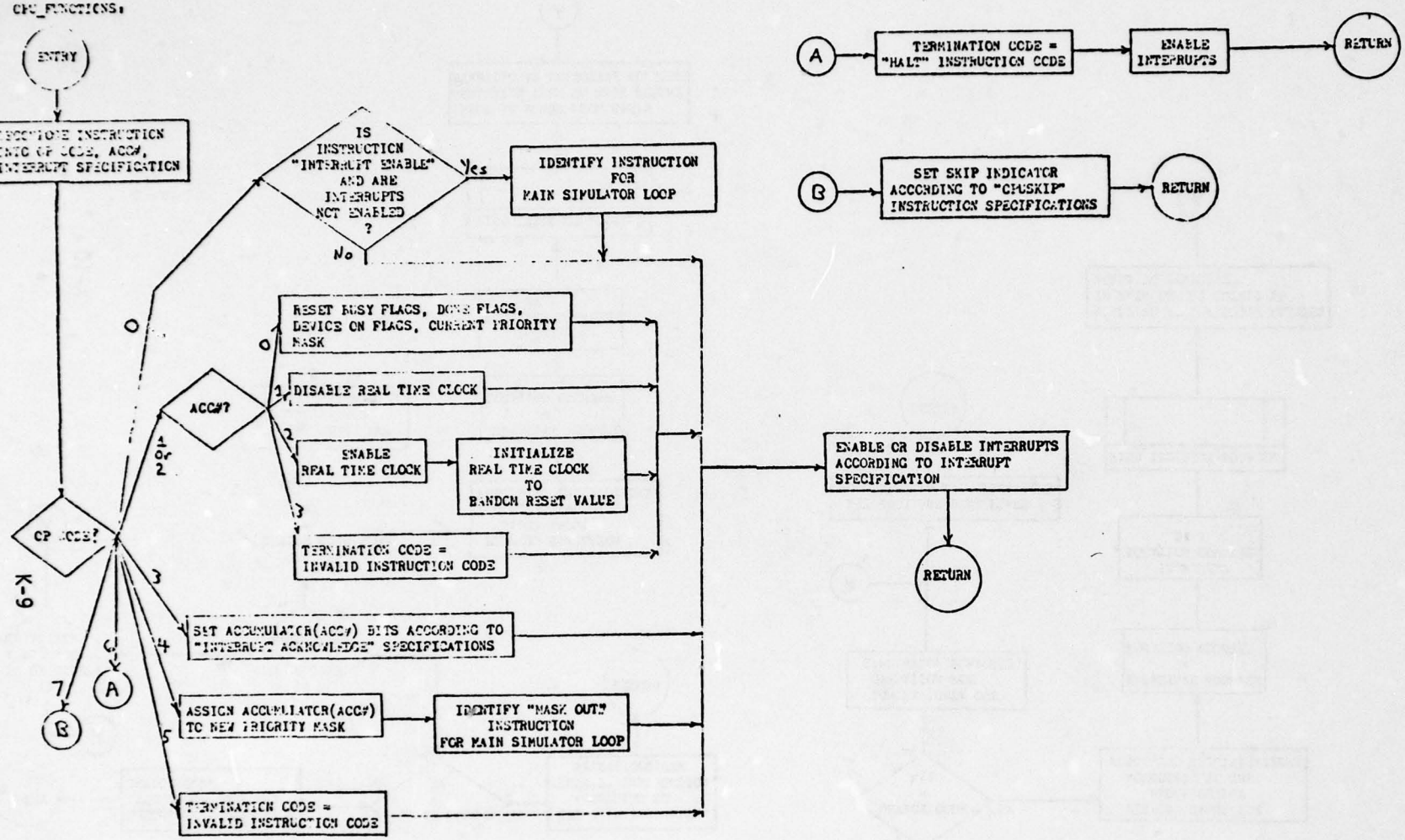

FIND_INDIRECT_ADDRESS:
4
ENTRY
INITIALIZE REFERENCE COUNT TO 2
RETURN
INCREMENT INSTRUCTION CYCLE COUNT BY INDIRECT ADDRESS CYCLE COUNT
REFERENCE COUNT ≤ 15?
Yes
No
TERMINATION CODE = INDIRECT OVERFLOW CODE
A
RETRIEVE INDIRECT WORD FROM MAIN MEMORY(EFFECTIVE ADDRESS)
16 ≤ EFFECTIVE ADDRESS ≤ 31
Yes
No
EFFECTIVE ADDRESS ≤ 23?
Yes
No
INCREMENT REFERENCE COUNT BY 2
A
INCREMENT INDIRECT WORD BY 1
DECREMENT INDIRECT WORD BY 1
LEFTMOST BIT OF INDIRECT WORD = 0?
Yes
No
STORE INDIRECT WORD IN MAIN MEMORY(EFFECTIVE ADDRESS)
RETURN
ASSIGN RIGHTMOST 15 BITS OF INDIRECT WORD TO EFFECTIVE ADDRESS
INCREMENT INSTRUCTION CYCLE COUNT BY AUTO INDEX CYCLE COUNT
REFERENCE COUNT > 15?
Yes
No
INCREMENT REFERENCE COUNT BY 1
TERMINATION CODE = INDIRECT OVERFLOW CODE

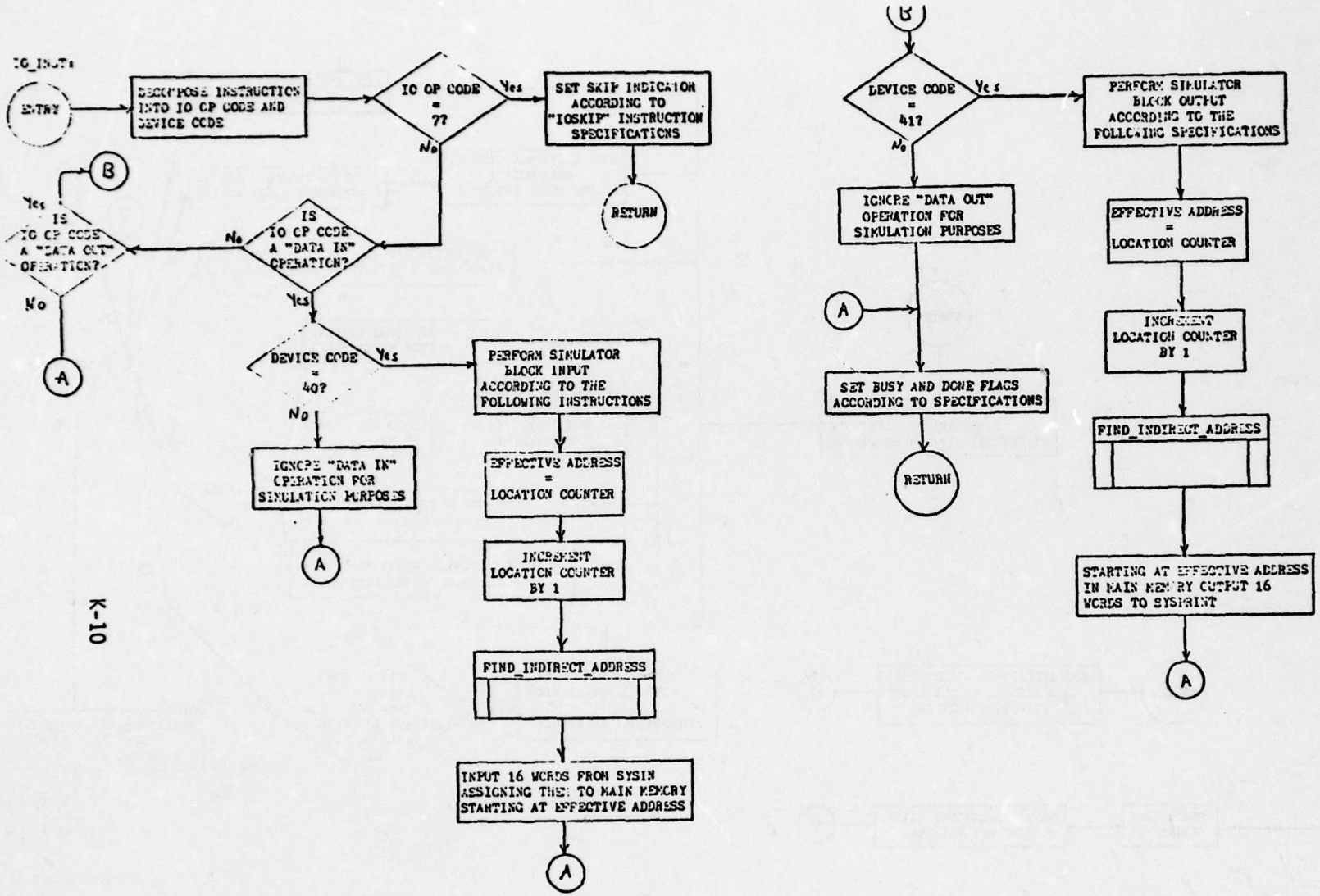

TWO_ACCUMULATOR_INSTR:
ENTRY
DECOMPOSE INSTRUCTION INTO SOURCE ACC#, DESTINATION ACC#, OP CODE, SHIFT ACTION, CARRY BIT ACTION, RESULT ACTION, AND SKIP ACTION
RESULT ACTION = 0 AND SKIP ACTION = 0?
Yes
No
TRAP_INST
CYCLE TABLE INDEX = 4
RETURN
SET CARRY BIT ACCORDING TO CARRY BIT ACTION
OPERAND = ACCUMULATOR(SOURCE ACC#)
PERFORM COMPLEMENT OF OPERAND WITH RESULT ASSIGNED TO SHIFTER
PERFORM TWO'S COMPLEMENT NEGATION OF OPERAND WITH RESULT ASSIGNED TO SHIFTER
ASSIGN OPERAND TO SHIFTER
ADD 1 TO OPERAND WITH RESULT ASSIGNED TO SHIFTER
ADD COMPLEMENT OF OPERAND TO ACCUMULATOR(DESTINATION ACC#) WITH RESULT ASSIGNED TO SHIFTER
SUBTRACT OPERAND FROM ACCUMULATOR(DESTINATION ACC#) WITH RESULT ASSIGNED TO SHIFTER
ADD OPERAND TO ACCUMULATOR(DESTINATION ACC#) WITH RESULT ASSIGNED TO SHIFTER
LOGICALLY "AND" OPERAND BITS WITH ACCUMULATOR(DESTINATION ACC#) BITS WITH RESULT ASSIGNED TO SHIFTER
OP CODE?
0
1
2
3
4
5
6
7
A
K-6
SET LEFTMOST BIT OF SHIFTER ACCORDING TO FUNCTIONAL SPECIFICATIONS
PERFORM SPECIFIED SHIFT ACTION ON SHIFTER
SET SKIP INDICATOR ACCORDING TO SKIP ACTION SPECIFICATIONS
RESULT ACTION = 1?
Yes
No
ACCUMULATOR(DESTINATION ACC#) = RIGHTMOST 16 BITS OF SHIFTER
SKIP ACTION = 0?
Yes
No
TERMINATION CODE = SIMULATOR FAILURE CODE
CYCLE TABLE INDEX = 1
RETURN
TRAP_INST:
ENTRY
ASSIGN LOCATION COUNTER TO MAIN MEMORY(38)
EFFECTIVE ADDRESS = 39
FIND_INDIRECT_ADDRESS
ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER
RETURN

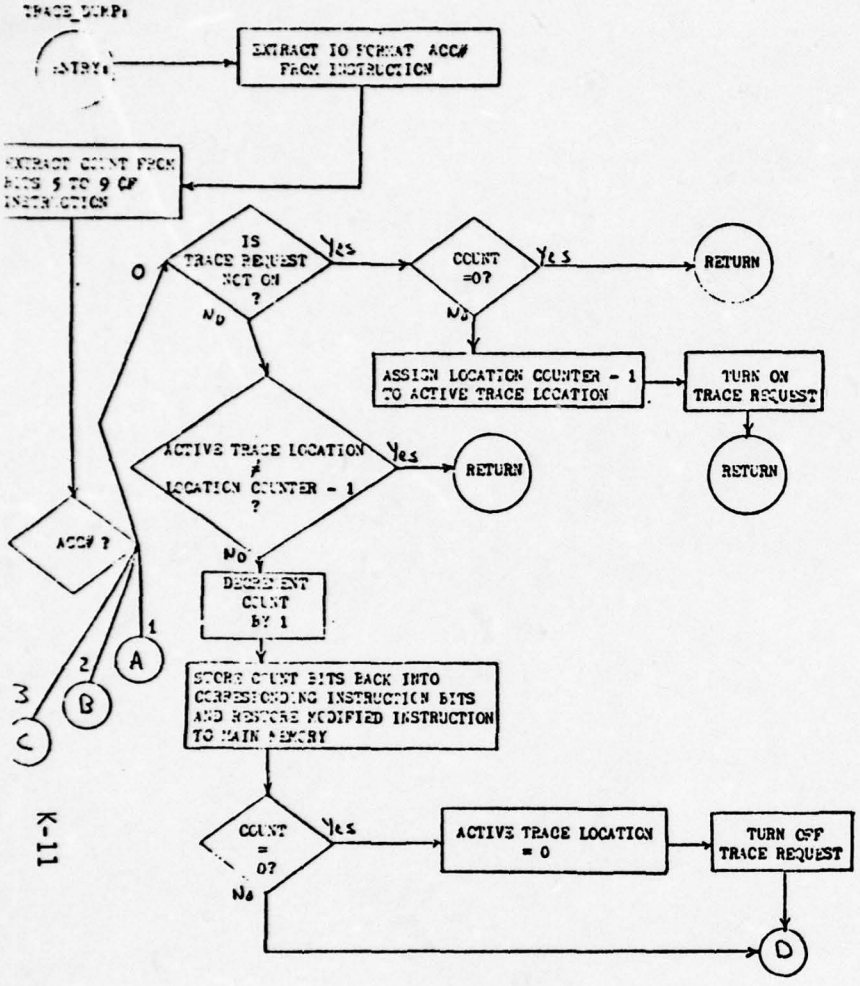

STACK_OR_MUL_DIV:
ENTRY
RETURN
OVERFLOW CODE = 1
CARRY BIT = '1'
MULTIPLY OR DIVIDE INSTRUCTION ?
Yes
No
DIVIDE INSTRUCTION ?
Yes
No
ACCUMULATOR(0) ≥ ACCUMULATOR(2) ?
Yes
No
PERFORM MULTIPLICATION OF ACCUMULATOR(1) BY ACCUMULATOR(2) WITH ACCUMULATOR(0) ADDED TO RESULT
ASSIGN RESULT TO ACCUMULATORS 0 AND 1
RETURN
PERFORM DIVISION OF CATENATED ACCUMULATORS 0 AND 1 BY ACCUMULATOR(2)
ASSIGN QUOTIENT AND REMAINDER TO ACCUMULATORS 0 AND 1, RESPECTIVELY
RETURN
DECONCTRUCT INSTRUCTION INTO STACK OP CODE AND ACC#
K-7
STACK OP CODE?
0
CYCLE TABLE INDEX = 1
ASSIGN ACCUMULATOR(ACC#) TO FRAME POINTER
1
CYCLE TABLE INDEX = 1
ASSIGN FRAME POINTER TO ACCUMULATOR(ACC#)
R
2
C2
3
C3
4
C4
5
C5
6
C6
7
C7

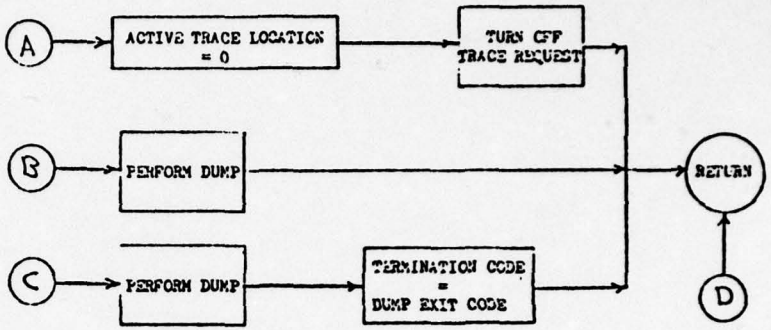

C2
INCREMENT STACK POINTER BY 1
LOOP = 0
LOOP = LOOP + 1
LOOP > 2?
Yes
No
CYCLE TABLE INDEX = 7
ASSIGN FRAME POINTER TO MAIN MEMORY(STACK POINT...
INCREMENT STACK POINTER BY 1
STACK OVERFLOW ?
Yes
No
SETUP STACK INTERRUPT REQUEST
RETURN
INCREMENT STACK POINTER BY 1
ASSIGN ACCUMULATOR(LOOP) TO MAIN MEMORY(STACK POINTER)
ASSIGN CARRY BIT AND ACCUMULATOR(3) TO MAIN MEMORY(STACK POINTER
ASSIGN STACK POINTER TO FRAME POINTER AND TO ACCUMULATOR(3)
R


2
CYCLE TABLE INDEX = 6
ASSIGN FRAME POINTER TO STACK POINTER
ASSIGN CARRY BIT AND LOCATION COUNTER FROM MAIN MEMORY(STACK POINTER)
4
CYCLE TABLE INDEX = 1
ASSIGN ACCUMULATOR(ACC#) TO STACK POINTER
DECREMENT STACK POINTER BY 1
5
CYCLE TABLE INDEX = 1
ASSIGN STACK POINTER TO ACCUMULATOR(ACC#)
LOOP = 3
LOOP = LOOP - 1
LOOP < 0?
Yes
No
ASSIGN MAIN MEMORY(STACK POINTER) TO ACCUMULATOR(LOOP)
DECREMENT STACK POINTER BY 1
6
CYCLE TABLE INDEX = 3
INCREMENT STACK POINTER BY 1
STACK OVERFLOW ?
Yes
No
SETUP STACK INTERRUPT
ASSIGN ACCUMULATOR(ACC#) TO MAIN MEMORY(STACK POINTER)
7
CYCLE TABLE INDEX = 3
ASSIGN MAIN MEMORY(STACK POINTER) TO ACCUMULATOR(ACC#)
DECREMENT STACK POINTER BY 1
RETURN
R


CPU_FUNCTIONS:
ENTRY
DECODE INSTRUCTION INTO OP CODE, ACC#, INTERRUPT SPECIFICATION
OP CODE?
K-9
IS INSTRUCTION "INTERRUPT ENABLE" AND ARE INTERRUPTS NOT ENABLED ?
Yes
No
IDENTIFY INSTRUCTION FOR MAIN SIMULATOR LOOP
0
1 or 2
ACC#?
RESET BUSY FLAGS, DONE FLAGS, DEVICE ON FLAGS, CURRENT PRIORITY MASK
DISABLE REAL TIME CLOCK
ENABLE REAL TIME CLOCK
INITIALIZE REAL TIME CLOCK TO RANDOM RESET VALUE
TERMINATION CODE = INVALID INSTRUCTION CODE
3
SET ACCUMULATOR(ACC#) BITS ACCORDING TO "INTERRUPT ACKNOWLEDGE" SPECIFICATIONS
4
ASSIGN ACCUMULATOR(ACC#) TO NEW PRIORITY MASK
IDENTIFY "MASK OUT" INSTRUCTION FOR MAIN SIMULATOR LOOP
5
TERMINATION CODE = INVALID INSTRUCTION CODE
6
A
7
B
ENABLE OR DISABLE INTERRUPTS ACCORDING TO INTERRUPT SPECIFICATION
RETURN
A
TERMINATION CODE = "HALT" INSTRUCTION CODE
ENABLE INTERRUPTS
RETURN
B
SET SKIP INDICATOR ACCORDING TO "CPUSKIP" INSTRUCTION SPECIFICATIONS
RETURN


IO_INST:
ENTRY
DECOMPOSE INSTRUCTION INTO IO OP CODE AND DEVICE CODE
IO OP CODE = 77
Yes
SET SKIP INDICATION ACCORDING TO "IOSKIP" INSTRUCTION SPECIFICATIONS
RETURN
No
IS IO OP CODE A "DATA IN" OPERATION?
No
Yes
IS IO OP CODE A "DATA OUT" OPERATION?
No
Yes
B
A
DEVICE CODE = 40?
Yes
No
PERFORM SIMULATOR BLOCK INPUT ACCORDING TO THE FOLLOWING INSTRUCTIONS
IGNORE "DATA IN" OPERATION FOR SIMULATION PURPOSES
EFFECTIVE ADDRESS = LOCATION COUNTER
INCREMENT LOCATION COUNTER BY 1
FIND_INDIRECT_ADDRESS
INPUT 16 WORDS FROM SYSIN ASSIGNING THEM TO MAIN MEMORY STARTING AT EFFECTIVE ADDRESS
B
DEVICE CODE = 41?
Yes
No
PERFORM SIMULATOR BLOCK OUTPUT ACCORDING TO THE FOLLOWING SPECIFICATIONS
IGNORE "DATA OUT" OPERATION FOR SIMULATION PURPOSES
EFFECTIVE ADDRESS = LOCATION COUNTER
INCREMENT LOCATION COUNTER BY 1
FIND_INDIRECT_ADDRESS
A
SET BUSY AND DONE FLAGS ACCORDING TO SPECIFICATIONS
STARTING AT EFFECTIVE ADDRESS IN MAIN MEMORY OUTPUT 16 WORDS TO SYSPRINT
RETURN


TRACE_DUMP:
ENTRY:
EXTRACT IO FORMAT ACC# FROM INSTRUCTION
EXTRACT COUNT FROM BITS 5 TO 9 OF INSTRUCTION
IS TRACE REQUEST NOT ON ?
Yes
No
COUNT =0?
Yes
No
RETURN
ASSIGN LOCATION COUNTER - 1 TO ACTIVE TRACE LOCATION
TURN ON TRACE REQUEST
RETURN
ACTIVE TRACE LOCATION ≠ LOCATION COUNTER - 1 ?
Yes
No
RETURN
ACC# ?
0
1
2
3
A
B
C
DECREMENT COUNT BY 1
STORE COUNT BITS BACK INTO CORRESPONDING INSTRUCTION BITS AND RESTORE MODIFIED INSTRUCTION TO MAIN MEMORY
COUNT = 0?
Yes
No
ACTIVE TRACE LOCATION = 0
TURN OFF TRACE REQUEST
D


A
ACTIVE TRACE LOCATION = 0
TURN OFF TRACE REQUEST
B
PERFORM DUMP
RETURN
C
PERFORM DUMP
TERMINATION CODE = DUMP EXIT CODE
D

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX L - SIMULATOR DESIGN, PDL FORM

Name_____

COMSC 5113 SIMULATOR DESIGN Spring 1977

Explanatory notes:

1) The location counter in this design generally points to the next instruction in contrast to the current instruction as stated in the microNova manual. This tends to simplify the design but has implications concerning tracing and dumping.

2) There are no deliberate errors in the design. If errors are detected or clarification of microNova functional characteristics occurs then revision information will be distributed in the form of memoranda.

3) Input/output devices are not fully simulated. A simulator block input and block output scheme is used instead.

4) Console operations are not simulated in the design.

5) Half of the class is receiving a PDL design and half of the class is receiving a flowchart design. These two designs contain equivalent information. If you are participating in the software design study you are requested to use only the design given you for your simulator implementation.

6) Questions concerning the design should be directed to Dr. J.R. Van Doren. Dr. D.D. Fisher should be consulted concerning questions about microNova functional characteristics.

```
MICRONOVA_SIMULATOR:
     PROCEDURE;
        CALL INITIALIZE_SIMULATOR;
        DO WHILE TERMINATION CODE = 0;
           FETCH INSTRUCTION;
           INCREMENT LOCATION COUNTER;
           IF PRIOR INSTRUCTION WAS "MASK OUT" THEN
              TURN ON PRIORITY MASK UPDATE SWITCH;
           CALL EXECUTE_INSTRUCTION;
           IF PRIORITY MASK UPDATE SWITCH IS ON THEN
              DO;
                 CURRENT PRIORITY MASK = NEW PRIORITY MASK;
                 TURN OFF PRIORITY MASK UPDATE SWITCH;
              END;
           INCREMENT CURRENT CYCLE COUNT BY INSTRUCTION CYCLE COUNT;
           IF CURRENT CYCLE COUNT > CYCLE LIMIT THEN
              TERMINATION CODE = EXCESSIVE CYCLE CODE;
           IF TRACE IS REQUESTED THEN
              DO;
                 PERFORM INSTRUCTION TRACE;
                 DECREMENT TRACE COUNT BY 1;
                 IF TRACE COUNT = 0 THEN TURN OFF TRACE REQUEST;
              END;
           IF SKIP INDICATOR IS ON THEN
              DO;
                 TURN OFF SKIP INDICATOR;
                 INCREMENT LOCATION COUNTER;
              END;
           IF REAL TIME CLOCK IS ENABLED THEN
              DO;
                 DECREMENT CLOCK CYCLE COUNT BY INSTRUCTION CYCLE COUNT;
                 IF CLOCK CYCLE COUNT≤0 THEN
                    DO;
                       SETUP INTERRUPT CONDITION;
                       RESET REAL TIME CLOCK;
                    END;
              END;
           INSTRUCTION CYCLE COUNT = 0;
           DO WHILE INTERRUPT CYCLE COUNT ≤ CURRENT CYCLE COUNT;
              IF DEVICE CODE IS ON AND CURRENT PRIORITY MASK ALLOWS
                 DEVICE INTERRUPT THEN SETUP INTERRUPT FROM INTERRUPT LIST;
              UPDATE INTERRUPT CYCLE COUNT TO NEXT INTERRUPT;
           END;
           IF AN INTERRUPT IS REQUESTED AND INTERRUPTS ARE ENABLED
              AND INSTRUCTION EXECUTED WAS NOT "INTERRUPT ENABLE"
              THEN CALL PROCESS_INTERRUPT;
        END;
        PRINT SIMULATOR STATUS INFORMATION UPON TERMINATION;
     END MICRONOVA_SIMULATOR;
```

```
INITIALIZE_SIMULATOR: PROCEDURE;
     LOAD MICRONOVA PROGRAM FROM CARD DECK;
     INITIALIZE LOCATION COUNTER AND MAIN MEMORY SIZE;
     INITIALIZE ALL INTERRUPT FLAGS, INPUT/OUTPUT FLAGS, PRIORITY MASKS,
     SIMULATOR CONTROL FLAGS, ETC.;
     INITIALIZE TERMINATION CODE, CURRENT CYCLE COUNT, INSTRUCTION CYCLE
     COUNT TO ZERO;
     INITIALIZE CYCLE COUNT TABLE AND CYCLE LIMIT;
     INITIALIZE  INTERRUPT LIST FOR INTERRUPT SIMULATION;
     INITIALIZE REAL TIME CLOCK RESET VALUE AND RANDOM RESET VALUE;
     PERFORM MISCELLANEOUS INTTIALIZATION;
     RETURN TO CALLER;
END INITIALIZE_SIMULATOR;


EXECUTE_INSTRUCTION: PROCEDURE;
     CYCLE TABLE INDEX = 1;
     IF LEFTMOST BIT OF INSTRUCTION IS '1' THEN CALL TWO_ACCUMULATOR_INST;
     ELSE IF NEXT TWO BITS OF INSTRUCTION ARE NOT '11' THEN
               CALL MEMORY_REFERENCE_INST;
               ELSE
                  DO;
                     ASSIGN RIGHTMOST SIX BITS OF INSTRUCTION TO DEVICE CODE;
                     IF DEVICE CODE = 1 THEN CALL STACK_OR_MUL_DIV;
                     ELSE IF DEVICE CODE = 62 THEN CALL TRACE_DUMP;
                          ELSE IF DEVICE CODE = 63 THEN CALL CPU_FUNCTION;
                               ELSE CALL IO_INST;
                  END;
     INCREMENT INSTRUCTION CYCLE COUNT BY CYCLE TABLE(CYCLE TABLE INDEX);
     RETURN TO CALLER;
END EXECUTE_INSTRUCTION;


PROCESS_INTERRUPT: PROCEDURE;
     DISABLE INTERRUPTS;
     INCREMENT INSTRUCTION CYCLE COUNT BY INTERRUPT CYCLE COUNT;
     STORE LOCATION COUNTER IN ZERO POSITION OF MAIN MEMORY VECTOR;
     DO REQUEST LINE = 3 TO 1 BY -1;
        IF INTERRUPT REQUEST(REQUEST LINE) IS ON THEN
           DO;
              EFFECTIVE ADDRESS = REQUEST LINE;
              CALL FIND_INDIRECT_ADDRESS;
              ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER;
              RETURN TO CALLER;
           END;
     END;
     TERMINATION CODE = INTERRUPT SIMULATION FAILURE CODE;
     RETURN TO CALLER;
END PROCESS_INTERRUPT;
```

```
MEMORY_REFERENCE_INST: PROCEDURE;
     CALL COMPUTE_EFFECTIVE_ADDRESS;
     IF BITS 1 AND 2 OF INSTRUCTION ARE '00' THEN CALL NO_ACCUMULATOR_INST;
     ELSE CALL ONE_ACCUMULATOR_INST;
     RETURN TO CALLER;
END MEMORY_REFERENCE_INST;


ONE_ACCUMULATOR_INST: PROCEDURE;
     EXTRACT OP CODE AND ACC#;
     DO CASE (OP CODE);
        CASE0:
           TERMINATION CODE = SIMILATOR FAILURE CODE;
        CASE1:
           DO;
              CYCLE TABLE INDEX = 2;
              LOAD ACCUMULATOR(ACC#) FROM MAIN MEMORY(EFFECTIVE ADDRESS);
           END;
        CASE2:
           DO;
              CYCLE TABLE INDEX = 3;
              STORE ACCUMULATOR(ACC#) IN MAIN MEMORY(EFFECTIVE ADDRESS);
           END;
        CASE3:
           TERMINATION CODE = SIMULATOR FAILURE CODE;
     END CASE;
     RETURN TO CALLER;
END ONE_ACCUMULATOR_INST;


NO_ACCUMULATOR_INST: PROCEDURE;
     EXTRACT OP CODE;
     DO CASE ( OP CODE );
        CASE0:
           DO;
              CYCLE TABLE INDEX = 2;
              ASSIGN EFFECTIVE ADDRESS RO LCCATION COUNTER;
           END;
        CASE1:
           DO;
              CYCLE TABLE INDEX = 3;
              ASSIGN LOCATION COUNTER TO ACCUMULATOR(3);
              ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER;
           END;
        CASE2:
           DO;
              CYCLE TABLE INDEX = 4;
              INCREMENT MAIN MEMORY(EFFECTIVE ADDRESS) BY 1;
              IF RESULT IS ZERO THEN TURN ON SKIP INDICATOR;
           END;
        CASE3:
           DO;
              CYCLE TABLE INDEX = 4;
              DECREMENT MAIN MEMORY(EFFECTIVE ADDRESS) BY 1;
              IF RESULT IS ZERO THEN TURN ON SKIP INDICATOR;
           END;
     END CASE;
     RETURN TO CALLER;
END NO_ACCUMULATOR_INST;
```

```
COMPUTE_EFFECTIVE_ADDRESS: PROCEDURE;
     DECOMPOSE 11 ADDRESS BITS INTO INDIRECT BIT, INDEXING CODE,
     SIGNED DISPLACEMENT, UNSIGNED DISPLACEMENT;
     DO CASE (INDEXING CODE);
        CASE0:
           EFFECTIVE ADDRESS = UNSIGNED DISPLACEMENT;
        CASE1:
           EFFECTIVE ADDRESS = LOCATION COUNTER - 1 + SIGNED DISPLACEMENT;
        CASE2:
           EFFECTIVE ADDRESS = ACCUMULATOR(2) + SIGNED DISPLACEMENT;
        CASE3:
           EFFECTIVE ADDRESS = ACCUMULATOR(3) + SIGNED DISPLACEMENT;
     END CASE;
     RETAIN RIGHTMOST 15 BITS OF EFFECTIVE ADDRESS;
     IF INDIRECT BIT IS ON THEN CALL FIND_INDIRECT_ADDRESS;
     RETURN TO CALLER;
END COMPUTE_EFFECTIVE_ADDRESS;


FIND_INDIRECT_ADDRESS: PROCEDURE;
     INITIALIZE REFERENCE COUNT TO 2;
     DO WHILE REFERENCE COUNT ≤ 15;
        INCREMENT INSTRUCTION CYCLE COUNT BY INDIRECT ADDRESS CYCLE COUNT;
        RETRIEVE INDIRECT WORD FROM MAIN MEMORY(EFFECTIVE ADDRESS);
        IF 16 ≤ EFFECTIVE ADDRESS ≤ 31 THEN
           DO;
              IF EFFECTIVE ADDRESS ≤ 23 THEN INCREMENT INDIRECT WORD BY 1;
              ELSE DECREMENT INDIRECT WORD BY 1;
              STORE INDIRECT WORD IN MAIN MEMORY(EFFECTIVE ADDRESS);
              INCREMENT INSTRUCTION CYCLE COUNT BY AUTO INDEX CYCLE COUNT;
              INCREMENT REFERENCE COUNT BY 1;
              IF REFERENCE COUNT > 15 THEN
                 DO;
                    TERMINATION CODE = INDIRECT OVERFLOW CODE;
                    RETURN TO CALLER;
                 END;
           END;
        ASSIGN RIGHTMOST 15 BITS OF INDIRECT WORD TO EFFECTIVE ADDRESS;
        IF LEFTMOST BIT OF INDIRECT WORD = 0 THEN RETURN TO CALLER;
        INCREMENT REFERENCE COUNT BY 2;
     END;
     TERMINATION CODE = INDIRECT OVERFLOW CODE;
     RETURN TO CALLER;
END FIND_INDIRECT_ADDRESS;
```

```
TWO_ACCUMULATOR_INSTR: PROCEDURE;
     DECOMPOSE INSTRUCTION INTO SOURCE ACC#, DESTINATION ACC#, OP CODE,
     SHIFT ACTION, CARRY BIT ACTION, RESULT ACTION, SKIP ACTION;
     IF RESULT ACTION = 0 AND SKIP ACTION = 0 THEN
        DO;
           CALL TRAP_INST;
           CYCLE TABLE INDEX = 4;
           RETURN TO CALLER;
        END;
     SET CARRY BIT ACCORDING TO CARRY BIT ACTION;
     OPERAND =ACCUMULATOR(SOURCE ACC#);
     DO CASE (OP CODE);
        CASE0:
           PERFORM COMPLEMENT OF OPERAND WITH RESULT ASSIGNED TO SHIFTER;
        CASE1:
           PERFORM TWO'S COMPLEMENT NEGATION OF OPERAND WITH RESULT
           ASSIGNED TO SHIFTER;
        CASE2:
           ASSIGN OPERAND TO SHIFTER;
        CASE3:
           ADD 1 TO OPERAND WITH RESULT ASSIGNED TO SHIFTER;
        CASE4:
           ADD COMPLEMENT OF OPERAND TO ACCUMULATOR(DESTINATION ACC#) WITH
           RESULT ASSIGNED TO SHIFTER;
        CASE5:
           SUBTRACT OPERAND FROM ACCUMULATOR(DESTINATION ACC#) WITH
           RESULT ASSIGNED TO SHIFTER;
        CASE6:
           ADD OPERAND TO ACCUMULATOR(DESTINATION ACC#) WITH RESULT
           ASSIGNED TO SHIFTER;
        CASE7:
           LOGICALLY "AND" OPERAND BITS WITH ACCUMULATOR(DESTINATION ACC#)
           BITS WITH RESULT ASSIGNED TO SHIFTER;
     END CASE;
     SET LEFTMOST BIT OF SHIFTER ACCORDING TO FUNCTIONAL SPECIFICATIONS;
     PERFORM SPECIFIED SHIFT ACTION ON SHIFTER;
     SET SKIP INDICATOR ACCORDING TO SKIP ACTION SPECIFICATIONS;
     IF RESULT ACTION = 1 THEN ACCUMULATOR(DESTINATION ACC#) =
                               RIGHTMOST 16 BITS OF SHIFTER;
     ELSE IF SKIP ACTION = 0 THEN TERMINATION CODE = SIMULATOR FAILURE CODE;
     CYCLE TABLE INDEX = 1;
     RETURN TO CALLER;
END_TWO_ACCUMULATOR_INSTR;


TRAP_INST: PROCEDURE;
     ASSIGN LOCATION COUNTER TO MAIN MEMORY(38);
     EFFECTIVE ADDRESS = 39;
     CALL FIND_INDIRECT_ADDRESS;
     ASSIGN EFFECTIVE ADDRESS TO LOCATION COUNTER;
     RETURN TO CALLER;
END TRAP_INST;
```

```
STACK_OR_MUL_DIV: PROCEDURE;
     IF MULTIPLY OR DIVIDE INSTRUCTION THEN
        DO;
           IF DIVIDE INSTRUCTION THEN
              DO;
                 IF ACCUMULATOR(0) ≥ ACCUMULATOR(2) THEN
                    DO;
                       CARRY BIT = 1;
                       OVERFLOW CODE = 1;
                       RETURN TO CALLER;
                    END;
                 PERFORM DIVISION OF CATENATED ACCUMULATORS 0 AND 1
                 BY ACCUMULATOR(2);
                 ASSIGN QUOTIENT AND REMAINDER TO ACCUMULATORS 0 AND 1,
                 RESPECTIVELY;
                 RETURN TO CALLER;
              END;
           PERFORM MULTIPLICATION OF ACCUMULATOR(1) BY ACCUMULATOR(2) WITH
           ACCUMULATOR(0) ADDED TO RESULT;
           ASSIGN RESULT TO ACCUMULATORS 0 AND 1;
           RETURN TO CALLER;
        END;
     DECOMPOSE INSTRUCTION INTO STACK OP CODE AND ACC#;
     DO CASE (STACK OP CODE);
        CASE0:
           DO;
              CYCLE TABLE INDEX = 1;
              ASSIGN ACCUMULATOR(ACC#) TO FRAME POINTER;
           END;
        CASE1:
           DO;
              CYCLE TABLE INDEX = 1;
              ASSIGN FRAME POINTER TO ACCUMULATOR(ACC#);
           END;
        CASE2:
          DO;
           DO LOOP = 0 TO 2 BY 1;
              INCREMENT STACK POINTER BY 1;
              IF STACK OVERFLOW THEN
                 DO;
                    SETUP STACK INTERRUPT REQUEST;
                    RETURN TO CALLER;
                 END;
              ASSIGN ACCUMULATOR(LOOP) TO MAIN MEMORY(STACK POINTER);
           END;
           INCREMENT STACK POINTER BY 1;
           CYCLE TABLE INDEX = 7;
           ASSIGN FRAME POINTER TO MAIN MEMORY(STACK POINTER);
           INCREMENT STACK POINTER BY 1;
           ASSIGN CARRY BIT AND ACCUMULATOR(3) TO MAIN MEMORY(STACK POINTER);
           ASSIGN STACK POINTER TO FRAME POINTER AND TO ACCUMULATOR(3);
          END;
```

```
      CASE3:
         DO;
            CYCLE TABLE INDEX = 6;
            ASSIGN FRAME POINTER TO STACK POINTER;
            ASSIGN CARRY BIT AND LOCATION COUNTER FROM MAIN MEMORY(STACK POINTER);
            DECREMENT STACK POINTER BY 1;
            DO LOOP = 3 TO 0 BY -1;
               ASSIGN MAIN MEMORY(STACK POINTER) TO ACCUMULATOR(LOOP);
               DECREMENT STACK POINTER BY 1;
            END;
         END;
      CASE4:
         DO;
            CYCLE TABLE INDEX = 1;
            ASSIGN ACCUMULATOR(ACC#) TO STACK POINTER;
         END;
      CASE5:
         DO;
            CYCLE TABLE INDEX = 1;
            ASSIGN STACK POINTER TO ACCUMMULATOR(ACC#);
         END;
      CASE6:
         DO;
            CYCLE TABLE INDEX = 3;
            INCREMENT STACK POINTER BY 1;
            IF STACK OVERFLOW THEN SETUP STACK INTERRUPT;
            ELSE ASSIGN ACCUMULATOR(ACC#) TO MAIN MEMORY(STACK POINTER);
         END;
      CASE7:
         DO;
            CYCLE TABLE INDEX = 3;
            ASSIGN MAIN MEMORY(STACK POINTER) TO ACCUMULATOR(ACC#);
            DECREMENT STACK POINTER BY 1;
         END;
   END CASE;
   RETURN TO CALLER;
END STACK_OR_MUL_DIV;
```

```
CPU_FUNCTIONS: PROCEDURE;
     DECOMPOSE INSTRUCTION INTO OP CODE, ACC#, INTERRUPT SPECIFICATION;
     DO CASE (OP CODE);
        CASE0:
           IF INSTRUCTION IS "INTERRUPT ENABLE" AND INTERRUPTS ARE NOT
              ENABLED THEN IDENTIFY INSTRUCTION FOR MAIN SIMULATOR LOOP;
        CASE1:
           EXECUTE "CASE2" CODE;
        CASE2:
           DO CASE (ACC#);
              INNER_CASE0:
                 RESET BUSY FLAGS, DONE FLAGS, DEVICE ON FLAGS,
                 CURRENT PRIORITY MASK;
              INNER_CASE1:
                 DISABLE REAL TIME CLOCK;
              INNER_CASE2:
                 DO;
                    ENABLE REAL TIME CLOCK;
                    INITIALIZE REAL TIME CLOCK TO RANDOM RESET VALUE;
                 END;
              INNER_CASE3:
                 TERMINATION CODE = INVALID INSTRUCTION CODE;
           END CASE;
        CASE3:
           SET ACCUMULATOR(ACC#) BITS ACCORDING TO "INTERRUPT ACKNOWLEDGE"
           SPECIFICATIONS;
        CASE4:
           DO;
              ASSIGN ACCUMULATOR(ACC#) TO NEW PRIORITY MASK;
              IDENTIFY "MASK OUT" INSTRUCTION FOR MAIN SIMULATOR LOOP;
           END;
        CASE5:
           TERMINATION CODE = INVALID INSTRUCTION CODE;
        CASE6:
           DO;
              TERMINATION CODE = "HALT" INSTRUCTION CODE;
              ENABLE INTERRUPTS;
              RETURN TO CALLER;
           END;
     END CASE;
     ENABLE OR DISABLE INTERRUPTS ACCORDING TO INTERRUPT SPECIFICATION;
     RETURN TO CALLER;
END CPU_FUNCTIONS;
```

```
IO_INST: PROCEDURE;
     DECOMPOSE INSTRUCTION INTO IO OP CODE AND DEVICE CODE;
     IF IO OP CODE = 7 THEN
        DO;
           SET SKIP INDICATOR ACCORDING TO "IOSKIP" INSTRUCTION SPECIFICATIONS;
           RETURN TO CALLER;
        END;
     IF IO OP CODE IS A "DATA IN" OPERATION THEN
        DO;
           IF DEVICE CODE = 40 THEN
              DO;
                 PERFORM SIMULATOR BLOCK INPUT ACCORDING TO THE
                 FOLLOWING INSTRUCTIONS;
                 EFFECTIVE ADDRESS = LOCATION COUNTER;
                 INCREMENT LOCATION COUNTER BY 1;
                 CALL FIND_INDIRECT_ADDRESS;
                 INPUT 16 WORDS FROM SYSIN ASSIGNING THEM TO MAIN MEMORY
                 STARTING AT EFFECTIVE ADDRESS;
              END;
           ELSE IGNORE "DATA IN OPERATION FOR SIMULATION PURPOSES;
        END;
     ELSE IF IO OP CODE IS A "DATA OUT" OPERATION THEN
              DO;
                 IF DEVICE CODE = 41 THEN
                    DO;
                       PERFORM SIMULATOR BLOCK OUTPUT ACCORDING TO THE
                       FOLLOWING INSTRUCTIONS;
                       EFFECTIVE ADDRESS = LOCATION COUNTER;
                       INCREMENT LOCATION COUNTER BY 1;
                       CALL FIND_INDIRECT_ADDRESS;
                       STARTING AT EFFECTIVE ADDRESS IN MAIN MEMORY
                       OUTPUT 16 WORDS TO SYSPRINT;
                    END;
                 ELSE IGNORE "DATA OUT" OPERATION FOR SIMULATION PURPOSES;
              END;
     SET BUSY AND DONE FLAGS ACCORDING TO SPECIFICATIONS;
     RETURN TO CALLER;
END IO_INST;
```

```
TRACE_DUMP: PROCEDURE;
     EXTRACT IO FORMAT ACC# FROM INSTRUCTION;
     EXTRACT COUNT FROM BITS 5 TO 9 OF INSTRUCTION;
     DO CASE (ACC#);
        CASE0:
           DO;
              IF TRACE REQUEST IS NOT ON THEN
                 DO;
                    IF COUNT = 0 THEN RETURN TO CALLER;
                    ASSIGN LOCATION COUNTER - 1 TO ACTIVE TRACE LOCATION;
                    TURN ON TRACE REQUEST;
                    TRACE COUNT = COUNT;
                    RETURN TO CALLER;
                 END;
              IF ACTIVE TRACE LOCATION ≠ LOCATION COUNTER - 1 THEN
                 RETURN TO CALLER;
              DECREMENT COUNT BY 1;
              STORE COUNT BITS BACK INTO CORRESPONDING INSTRUCTION BITS
              AND RESTORE MODIFIED INSTRUCTION TO MAIN MEMORY;
              IF COUNT = 0 THEN
                 DO;  ACTIVE TRACE LOCATION = 0;  TURN OFF TRACE REQUEST;  END;
           END;
        CASE1:
           DO;  ACTIVE TRACE LOCATION = 0;  TURN OFF TRACE REQUEST;  END;

        CASE2:
           PERFORM DUMP;
        CASE3:
           DO;  PERFORM DUMP;  TERMINATION CODE = DUMP EXIT CODE;  END;
     END CASE;
     RETURN TO CALLER;
END TRACE_DUMP;
```

APPENDIX M - SIMULATOR DESIGN REVISION MEMO

Errors and corrections in the simulator design. (both versions)

1) STACK_OR_MUL_DIV module (page 6)
The relational operator of the division overflow test is missing or incomplete
It should be
ACCUMULATOR(0) ≤ ACCUMULATOR(2).

2) STACK_OR_MUL_DIV module (page 6)
the accumulator numbers for the placement of quotient and remainder results for division are reversed.

3) STACK_OR_MUL_DIV module (page 6a)
The stack overflow test should be performed after stacking is completed

4) TWO_ACCUMULATOR_INSTR (page 5)
The result action bit test for detecting a TRAP instruction is inverted

5) INITIALIZE_SIMULATOR (page 2).
Initialization of main memory size should be performed before "loading" the microNova program.

APPENDIX N - DESIGN COMPREHENSION TEST

Name: _____

QUIZ

This quiz consists of thirty multiple-choice questions and two fill-in questions. Please pace yourself so that you can complete all questions in the time allowed. Answer all questions. If you don't know the answer to a multiple-choice question, eliminate any responses which are clearly wrong, and guess among the remaining responses.

Code: _____

DO NOT TURN THE PAGE until you have completed the work on this page. Then do not return to this page.

List below the names of all the modules in the simulator. List them exactly if you can; if you are unable to reproduce a module name exactly, then come as close as you can.

For each of the multiple-choice questions which follow, select the one best alternative by circling the letter which precedes it.

Example: Most automobiles are powered by
(a) steam engines.
(b) turbine engines.
(c) internal-combustion engines.
(d) horses.

1. According to the design, a simulator entity called DEVICE CODE

(a) is used to identify a peripheral device for I/O instructions.
(b) is used to prevent some instruction classes from being interpreted as I/O instructions.
(c) both (a) and (b).
(d) none of these.

2. Assume that the microNova simulator has been implemented in accordance with the design you have been given, and that you are now debugging the simulator. You observe that the "Load Accumulator" instruction works correctly, but the "Store Accumulator" instruction does not. Which module probably contains the error?

(a) EXECUTE_INSTRUCTION
(b) CPU_FUNCTIONS
(c) COMPUTE_EFFECTIVE_ADDRESS
(d) MEMORY_REFERENCE_INST

3. According to the design, the location counter is incremented

(a) immediately before instruction fetching.
(b) immediately after instruction fetching.
(c) immediately after instruction execution.
(d) if and only if the skip indicator is on.

4. The multiway branch specifications used in the design specify for an "out of range" condition that

(a) a precise, but variable, action is always to be taken.
(b) nothing is to be done as a consequence.
(c) an error message is to be printed and execution continued.
(d) the termination code is to be set to an "abort" code.

5. How many input/output device codes does the microNova allow?

(a) 16
(b) 32
(c) 64
(d) 128

6. Suppose the MEMORY_REFERENCE_INST module is invoked immediately upon entry to the EXECUTE_INSTRUCTION module. (Assume the invocation in the current design is removed.) The implications are that

   (a) redundant processing will occur but logical equivalence with the current design results.
   (b) logical equivalence will result with no redundant processing.
   (c) the CYCLE TABLE INDEX value may be incorrectly set.
   (d) incorrect simulation may result.

7. The manner in which the current cycle count is used in the simulator will

   (a) prevent infinite loops in the simulator program.
   (b) prevent infinite loops in the microNova program.
   (c) both (a) and (b).
   (d) none of these.

8. Suppose the simulator will eventually be used to simulate microNova programs in object deck form whose format is not yet known. In order to simulate these programs

   (a) the instruction execution module will have to be redesigned.
   (b) simulation of interrupts and I/O will need revision.
   (c) simulator initialization will require revision.
   (d) the simulator main procedure will need revision.

9. If the specifications for FETCH INSTRUCTION and invocation of EXECUTE_INSTRUCTION are interchanged, the implications are that

   (a) logical equivalence with the current design will result.
   (b) logical equivalence with the current design will result if simulator initialization is modified to fetch the first instruction.
   (c) logical equivalence with the current design will result if all location counter value references are changed and simulator initialization is modified to fetch the first instruction.
   (d) logical equivalence with the current design will require substantial redesign of several modules.

10. Suppose you are debugging the simulator, and you observe that you can successfully turn the trace on, that trace information is correctly printed, but that you cannot turn the trace off. Which module probably contains the error?

    (a) MICRONOVA_SIMULATOR (main procedure)
    (b) EXECUTE_INSTRUCTION
    (c) TRACE_DUMP
    (d) IO_INST

11. In the interrupt processing procedure there is a section of logic for assigning a failure code to the termination code. If this logic is executed the meaning is that

    (a) there is an error in the microNova program.
    (b) there is an error in the indirect address computation.
    (c) there is an error in the simulator program.
    (d) none of these.

12. The failure code reference in question 11 represents

    (a) a simulation of a microNova hardware feature.
    (b) a simulation of a feature in the microNova interrupt service program.
    (c) a feature of the simulation design.
    (d) all of the above.

13. The current cycle count represents

    (a) the number of microNova instructions simulated.
    (b) the index of the main simulator loop.
    (c) an approximation to the time of execution of the simulator program.
    (d) an approximation to the time of execution of the microNova program.

14. In microNova instructions which involve relative addressing, a part of the instruction word contains displacement information. How is that information represented?

    (a) one's complement
    (b) two's complement
    (c) excess-64
    (d) excess 128

15. If an effective address is computed which is outside the range of addresses of the machine being simulated, the design

    (a) clearly specifies that "wraparound" addressing results.
    (b) specifies that an "abort" termination code is to be set.
    (c) is not very precise about the action to be taken.
    (d) none of these.

16. According to the design, the procedure labelled FIND_INDIRECT_ADDRESS

(a) may be used to follow a chain of indirect addresses.
(b) is always used to retrieve the contents of exactly one word; thus a chain of indirect addresses is processed by repeated invocation of the named procedure.
(c) may not be invoked until address modification due to indexing (if any) has been performed.
(d) could easily be absorbed into the design of the module for memory reference instructions.

17. The cycle table index is used for

(a) indexing a table used for recording statistics about the simulation process.
(b) retrieving approximate instruction execution times.
(c) a loop index during cycle table processing.
(d) none of these.

18. The index bits for memory reference instructions, interpreted as a nonnegative integer value are used in the simulator design

(a) as an index to a vector of size four used for simulating the accumulators used as index registers.
(b) to modify the signed displacement.
(c) as a decrement value for auto index word references.
(d) to determine an "intermediate" effective address just before possible modification due to indirect addressing.

19. If the skip indicator is removed from the design, then it will be necessary to make which one of the following additional changes in the design?

(a) immediately after returning from the instruction execution routine, the instruction just executed should be tested to determine whether the location counter is to be incremented.
(b) modify the design for simulation of the JUMP instruction.
(c) whenever an instruction is fetched, it should be tested immediately for a skip specification; if one is present, the location counter should be incremented.
(d) wherever the skip indicator is turned on in the current design, replace that with a location counter increment.

20. Suppose you are debugging the simulator, and you observe that "Trap" instructions work correctly, and that "Load Accumulator" and "Jump" instructions do not work when they use indirect addressing, but succeed otherwise. Which module probably contains the error?

(a) FIND_INDIRECT_ADDRESS
(b) COMPUTE_EFFECTIVE_ADDRESS
(c) MEMORY_REFERENCE_INST
(d) EXECUTE_INSTRUCTION

21. For purposes of illustration, suppose there are 25 instructions in the microNova instruction set. An alternative way of designing the simulator might be to provide a procedure to determine which one of the 25 instructions is to be simulated and then have a 25-way branch to 25 different sets of code which perform the actual simulation. Compared to the design given to you, the implications of such an approach are that

    (a) substantial duplication of code might result unless such duplication is removed to subprograms.
    (b) modification of the resulting simulator would be simpler, although initial implementation would take longer.
    (c) execution time for the simulator would be significantly shorter.
    (d) none of these.

22. The purpose of the CPU_FUNCTIONS module is to

    (a) assign values to CPU registers from a special input device.
    (b) simulate instructions associated with interrupts.
    (c) simulate the microNova CPU.
    (d) none of these.

23. The outermost loop of the simulator is controlled by

    (a) the instruction cycle count.
    (b) pending interrupts.
    (c) a loop index.
    (d) a termination code.

24. The tracing and dumping features may be used for

    (a) debugging microNova programs.
    (b) debugging the simulator.
    (c) both (a) and (b).
    (d) none of these.

25. If an interrupt is to be simulated, the interrupt processing procedure

    (a) fetches the first instruction of the interrupt service routine.
    (b) decrements a value in simulated memory by one.
    (c) invokes the procedure for computation of effective addresses for proper determination of the location counter value.
    (d) none of the above.

26. The function of the microNova shifter, when executing a left shift operation, can best be described as a

    (a) 16-bit end-around shift.
    (b) 16-bit no-end-around shift.
    (c) 17-bit end-around shift.
    (d) 17-bit no-end-around shift.

27. Suppose the design is to be revised so the body of the outer loop of the simulator is shorter but the entire design is logically equivalent to the original design. Without changing any other procedures, it would be possible to put some of the processing in

    (a) the simulator initialization module.
    (b) the instruction execution module.
    (c) the module which computes effective addresses.
    (d) any of the above.

28. The purpose of the simulator initialization procedure is to

    (a) load a microNova machine language program into simulated memory.
    (b) initialize simulated machine flags and registers.
    (c) initialize simulator program control variables.
    (d) all of the above.

29. Suppose you are debugging the simulator, and you observe that "Interrupt Enable" instructions do not work correctly. Which module probably contains the error?

    (a) CPU_FUNCTIONS
    (b) PROCESS_INTERRUPT
    (c) IO_INST
    (d) EXECUTE_INSTRUCTION

30. The maximum number of microNova instructions simulated in one pass through the body of the outermost simulator loop is

    (a) 1.
    (b) 2.
    (c) 3.
    (d) 4 or more.

Shown on the next page is an adjacency matrix. The matrix is to be used to indicate which modules invoke which other modules directly. Rows represent calling modules, while columns represent called modules. You are to enter an "X" in each cell which represents a calling-module/called-module relation in the simulator design. Be sure you understand the example below before you fill in the matrix on the next page.

Example: Suppose that a design contains modules A, B, C, and D. Suppose that A calls B and C, and that B calls C and D. Then the matrix for this design has the form:

Calling module \ Called module	A	B	C	D
A		X	X	
B			X	X
C				
D				

Calling module \ Called module	TRACE_DUMP	INITIALIZE_SIMULATOR	COMPUTE_EFFECTIVE_ADDRESS	IO_INST	MICRONOVA_SIMULATOR	MEMORY_REFERENCE_INST	FIND_INDIRECT_ADDRESS	TRAP_INST	CPU_FUNCTIONS	EXECUTE_INSTRUCTION	STACK_OR_MUL_DIV	TWO_ACCUMULATOR_INSTR	PROCESS_INTERRUPT
TRACE_DUMP													
INITIALIZE_SIMULATOR													
COMPUTE_EFFECTIVE_ADDRESS													
IO_INST													
MICRONOVA_SIMULATOR													
MEMORY_REFERENCE_INST													
FIND_INDIRECT_ADDRESS													
TRAP_INST													
CPU_FUNCTIONS													
EXECUTE_INSTRUCTION													
STACK_OR_MUL_DIV													
TWO_ACCUMULATOR_INSTR													
PROCESS_INTERRUPT													

APPENDIX O - TIME EXPENDITURE QUESTIONNAIRE, IMPLEMENTATION PHASE

WEEKLY TIME QUESTIONNAIRE (Implementation Phase)

Name:_____

Date:_____ Code:_____

Please record the time you have spent on your class project in the last week, excluding time spent in class. Record time in hours and/or minutes by category.

Time spent in:	hrs : min
Initial code preparation	___:___
Modification of existing code	___:___
Keypunching	___:___
Preparation of test cases	___:___
Execution and interpretation of test runs	___:___
Preparation of documentation	___:___
Just thinking about project	___:___
Other (indicate nature of activity)	
_____	___:___
_____	___:___

This information is solely for experimental purposes; your responses will in no way affect your course grade. Please be as accurate as possible.

"Average quality" design from Group A, Pass 1, PDL condition

PASS I A14

Description of Procedures (Assembler)

PASS_ONE	pass one of the assembler described for SCAL
DECODE	used to decode instructions and aid PASS_ONE assemble the instructions
COPY	a procedure to copy the assembled and input image to a file for future use in PASS_TWO
PASS_TWO	pass two of the assembler described for SCAL
OP_CODE_TABLE	this procedure does a table look up and returns the value specified below for each op code

STORE	1	WRITE	13	DC	20
LOAD	2	READ	14		
SUB	5	DIO	0		
ADD	6	HALT	0		
BRO	8	PROG	16		
BR1	9	EQUATE	17		
BR2	10	LOC	18		
BR3	11	DS	19		

TABLE	a procedure to build the symbol (table)
CHECK	a procedure to check for multiple symbol table entries

Procedure variables:

COUNTER	the location counter
CODE	the operation code of the input instruction
TEMP	value returned by OP_CODE_TABLE
IMAGE	the image of the assembled instruction
ERR	the error field of the instruction

0 - no error
1 - invalid OP code
2 - missing end statement
3 - misplaced PROG statement
4 - missing label
5 - memory overflow

LOC	the label in the input image
ADDR	the address field of the input image
MEMORY_SIZE	the max address of addressable storage

Comments:

1. If a PROG statement does not appear first the location counter is set to 0. A PROG appearing anywhere else in the program is considered to be an error

```
PASS_ONE: PROCEDURE;
    COUNTER = 0;
    DO UNTIL ALL INSTRUCTIONS ARE PROCESSED;
        GET NEXT INSTRUCTION;
        IF CODE = END THEN DO;
            CALL CHECK
            CALL PASS_TWO;
            END;
        ELSE DO;
            CALL DECODE(CODE);
            CALL COPY;
            END;
    END;
END PASS_ONE;

PROCEDURE CHECK
    SORT THE SYMBOL TABLE INTO ALPHABETIC ORDER;
    CHECK TO SEE IF ANY TWO CONSECUTIVE ENTRIES ARE THE SAME;
END CHECK;
```

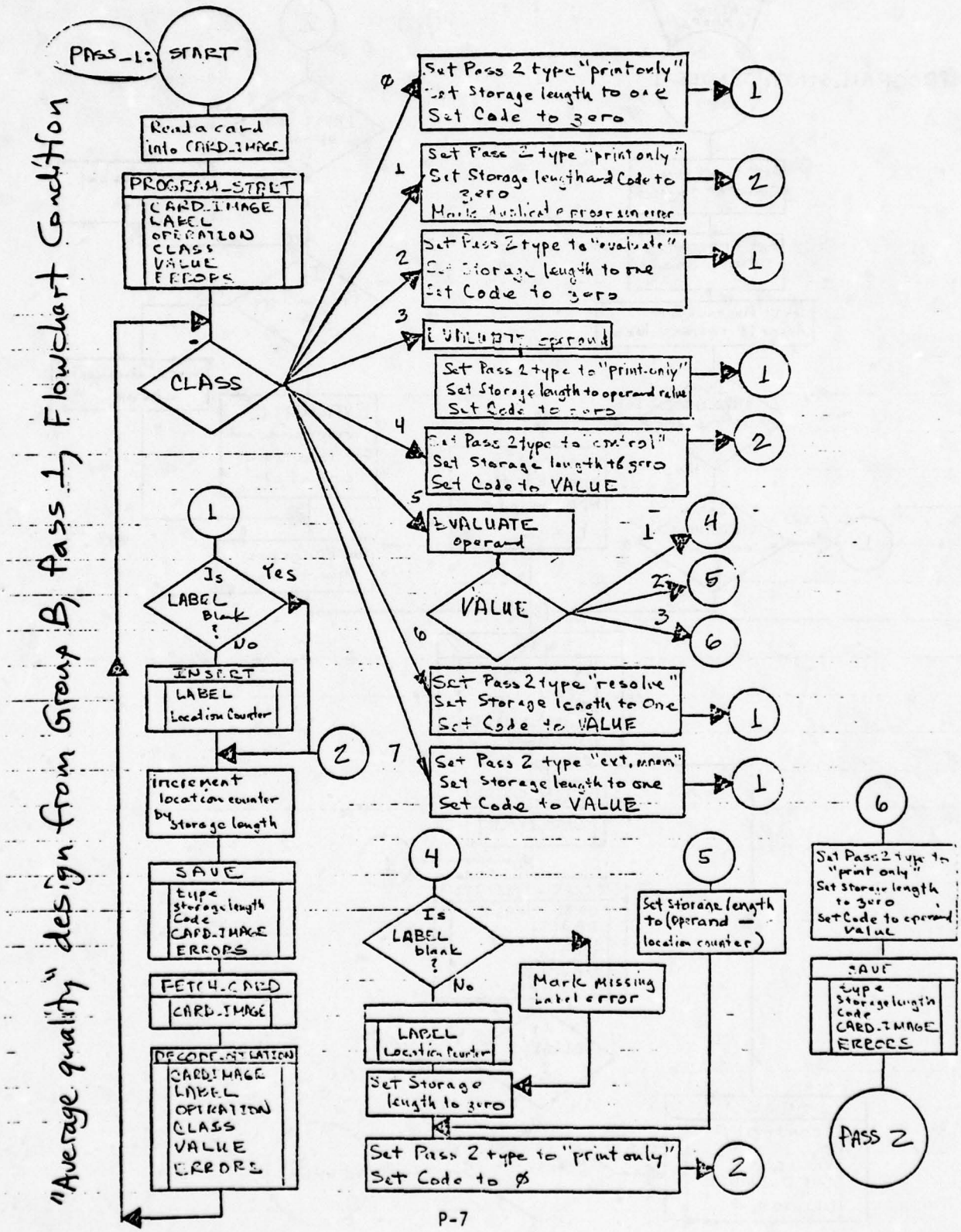
```
DECODE :PROCEDURE (CODE) ;
    CALL OP_CODE_TABLE (CODE, TEMP) ;
    IMAGE= IMAGE * 4096 ;
    IF CODE < 0 THEN DO;
        IMAGE = 0
        ERR = 1
        END;
    DO CASE TEMP ;
    CASE 16 : DO ;
            IF COUNTER ≠ 0 THEN DO;
                IMAGE=0
                ERR= 3
                END;
            ELSE IF LOC > MEMORY_SIZE THEN DO;
                ERR = 5 ;
                IMAGE=0;
                END ;
            ELSE COUNTER = LOC ;
            RETURN
    END ;
```

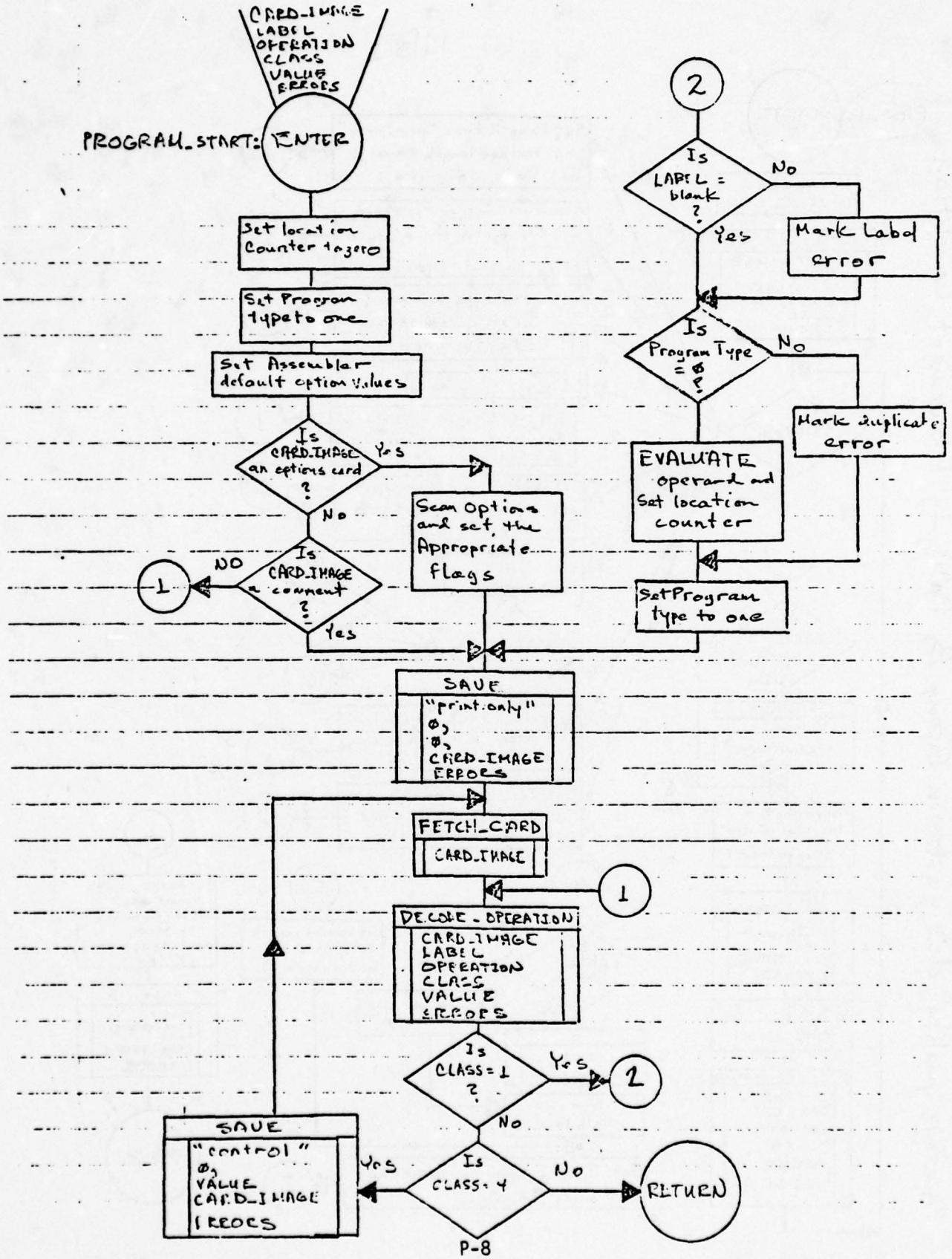
```
CASE 17 : DO;
        IF LOC = BLANK THEN DO;
            IMAGE = 0;
            ERR = 4;
            END;
        ELSE CALL TABLE(LOC, COUNTER);
        RETURN;
    END;
CASE 18 : DO;
        IF LOC > MEMORY_SIZE THEN ERR = 5
        ELSE COUNTER = LOC;
        RETURN;
    END;
CASE 19 : DO;
        IF LOC ¬= BLANK THEN CALL TABLE (LOC, COUNTER);
        COUNTER = COUNTER + ADDR;
        IF COUNTER > MEMORY_SIZE THEN ERR = 5;
        IMAGE = 0
        RETURN
    END;
```

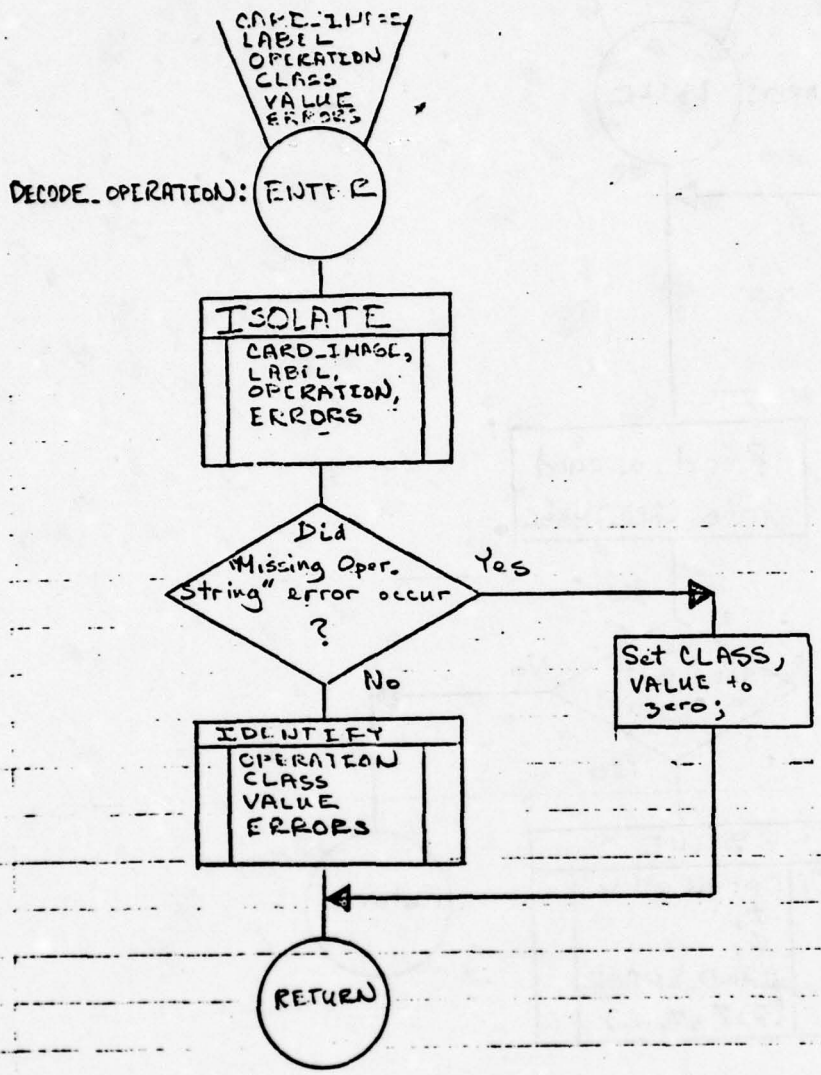
```
        CASE 20: DO;
                IF ADDR > MEMORY_SIZE THEN ERR = 5;
                IMAGE = ADDR;
                END;
        END;
IF LOC ≠ BLANK THEN CALL TABLE(LOC, COUNTER);
COUNTER = COUNTER + 1
IF COUNTER > MEMORY_SIZE THEN ERR = 5;
RETURN;
END DECODE;
```

BB1

"Average quality" design from Group B, Pass 1 Flowchart Condition



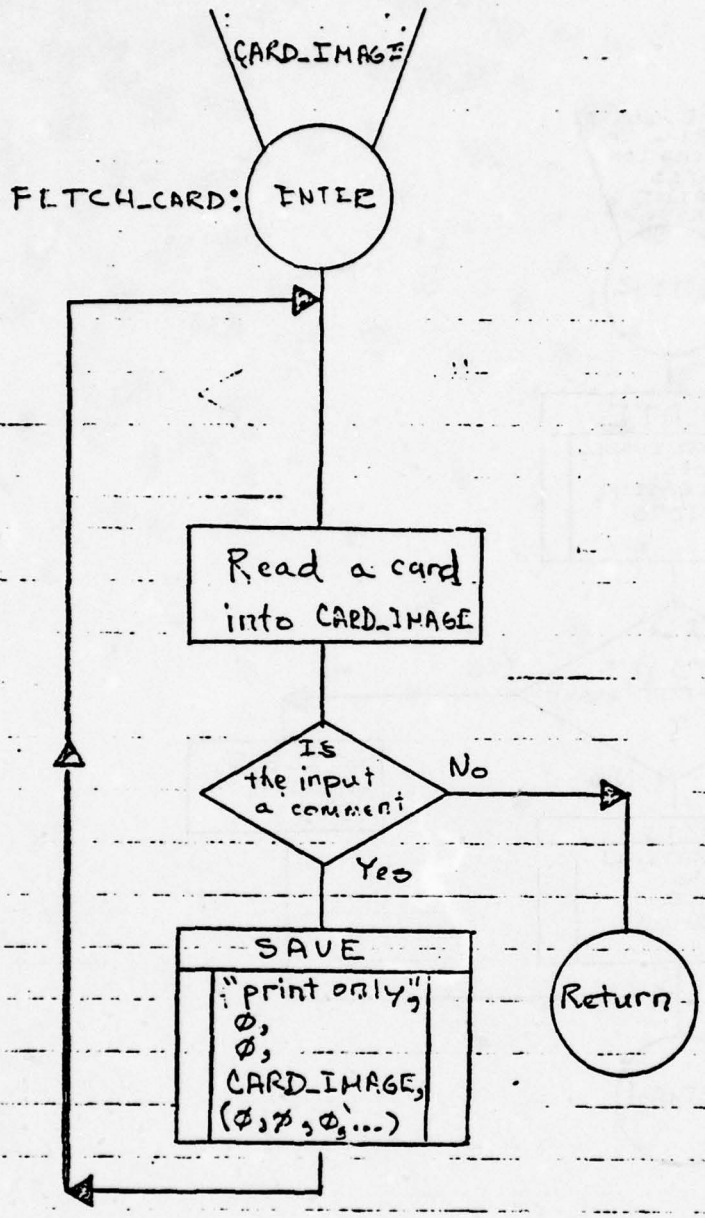

CARD_IMAGE
LABEL
OPERATION
CLASS
VALUE
ERRORS
PROGRAM_START: ENTER
Set location Counter to zero
Set Program type to one
Set Assembler default option values
Is CARD_IMAGE an options card ?
Yes
No
Scan Options and set the Appropriate flags
Is CARD_IMAGE a comment ?
NO
1
Yes
SAVE
"print only"
0,
0,
CARD_IMAGE
ERRORS
FETCH_CARD
CARD_IMAGE
1
DECODE_OPERATION
CARD_IMAGE
LABEL
OPERATION
CLASS
VALUE
ERRORS
Is CLASS = 1 ?
Yes
2
No
SAVE
"control"
0,
VALUE
CARD_IMAGE
ERRORS
Yes
Is CLASS = 4
No
RETURN
2
Is LABEL = blank ?
No
Yes
Mark Label error
Is Program Type = 0 ?
No
Mark duplicate error
EVALUATE operand and Set location counter
Set Program type to one



operation string classifications

Class	Op. Mnem. (VALUE)
1	PROG
2	DC, DID
3	DS
4	LIST(1), TITLE(2), PAGE(3), LINE(4)
5	EQUATE(1), LOC(2), END(3)
6	HALT(0200), STORE(1400), LOAD(2000), SUB(5200), ADD(6200), BR( ), WRITE(A200), READ(E200)
7	BOVFL(1), BRBSY(2), BWBSY(3)
Ø	UNDEFINED

Base 16 values


CARD_IMAGE
FETCH_CARD:
ENTRY
Read a card into CARD_IMAGE
Is the input a comment
No
Yes
SAVE
"print only",
Ø,
Ø,
CARD_IMAGE,
(Ø,Ø,Ø,...)
Return

Primitives

1) SAVE - save code and card image for Pass 2
   Input - Pass 2 classification
         - storage length
         - Instruction code / Pass 2 operation code
         - card image
         - Pass 1 error codes (vector)

2) INSERT - insert symbol into symbol table
   Input - Symbol
         - Value
   Output - Multiply def. error

3) RETRIEVE - retrieve symbol value from symbol table
   Input - Symbol
   Output - Value
          - Undefined error

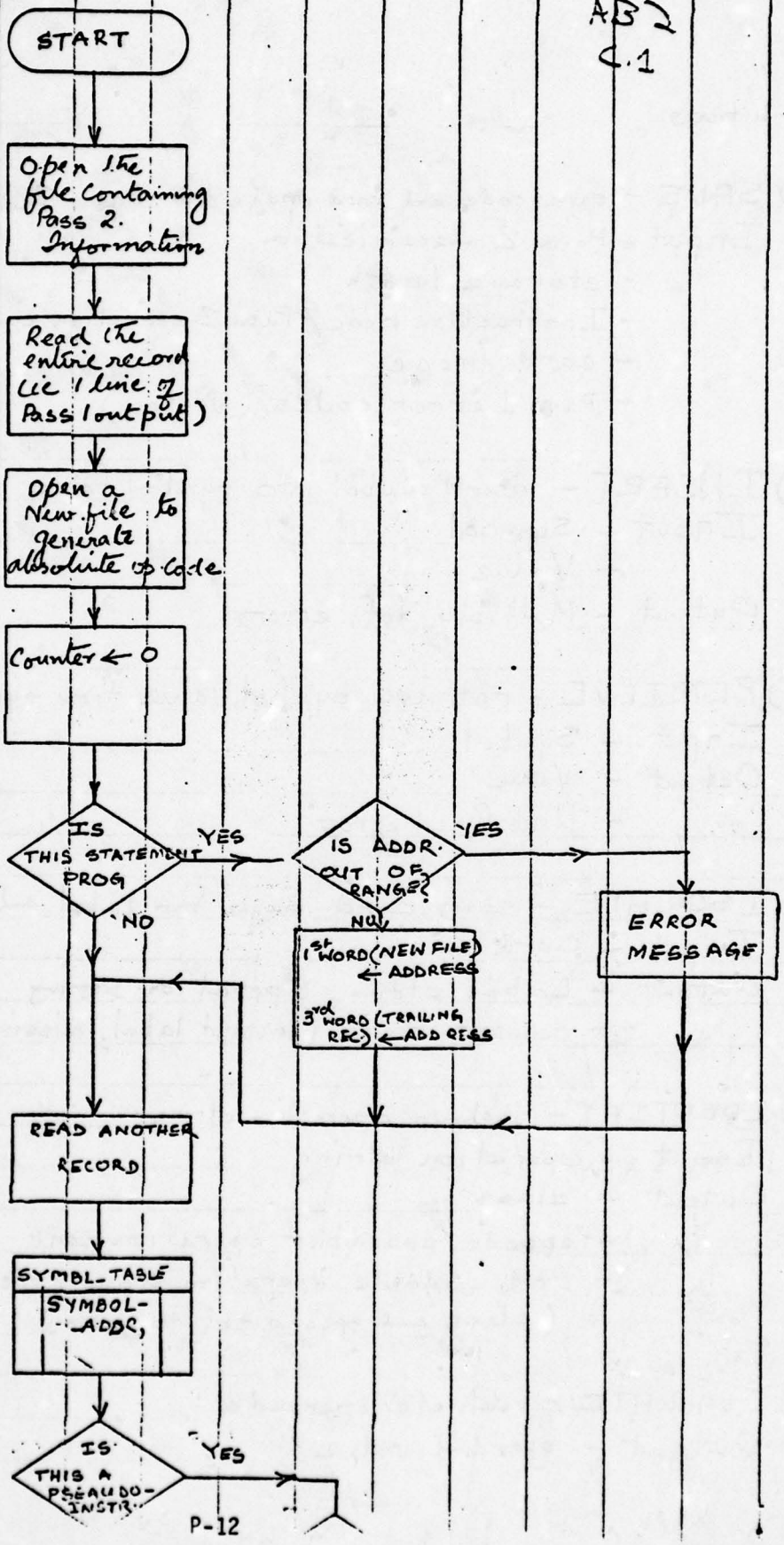
4) ISOLATE - scan card-image for label and operation string
   Input - Card image
   Output - Label string, Operation string
          - scanner errors (invalid label, missing operation string)

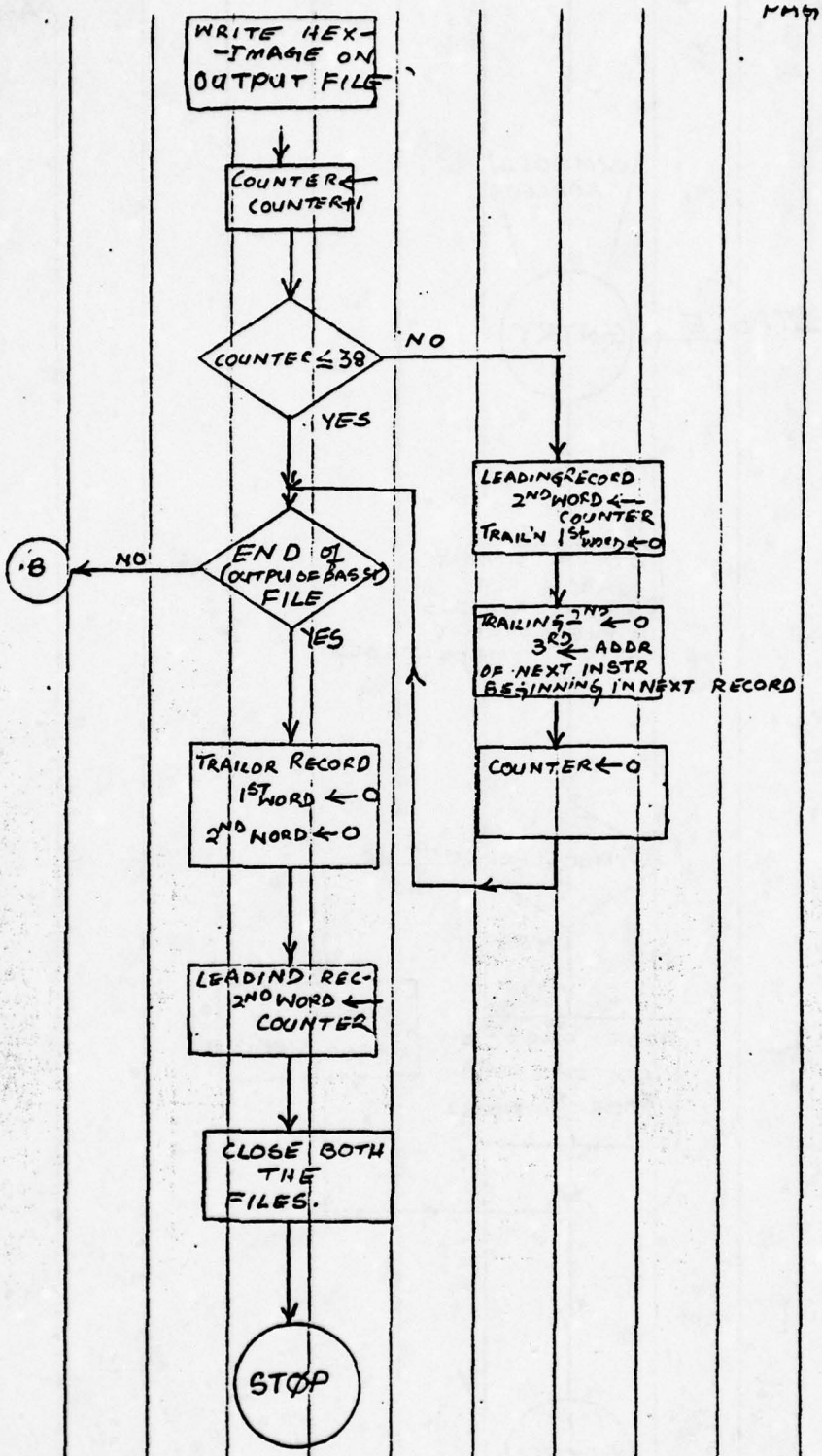
5) IDENTIFY - look-up operation string
   Input - operation string
   Output - class
          - opcode/assembler operation code
          - undecodable operation string error
            (class and opcode set to zero)

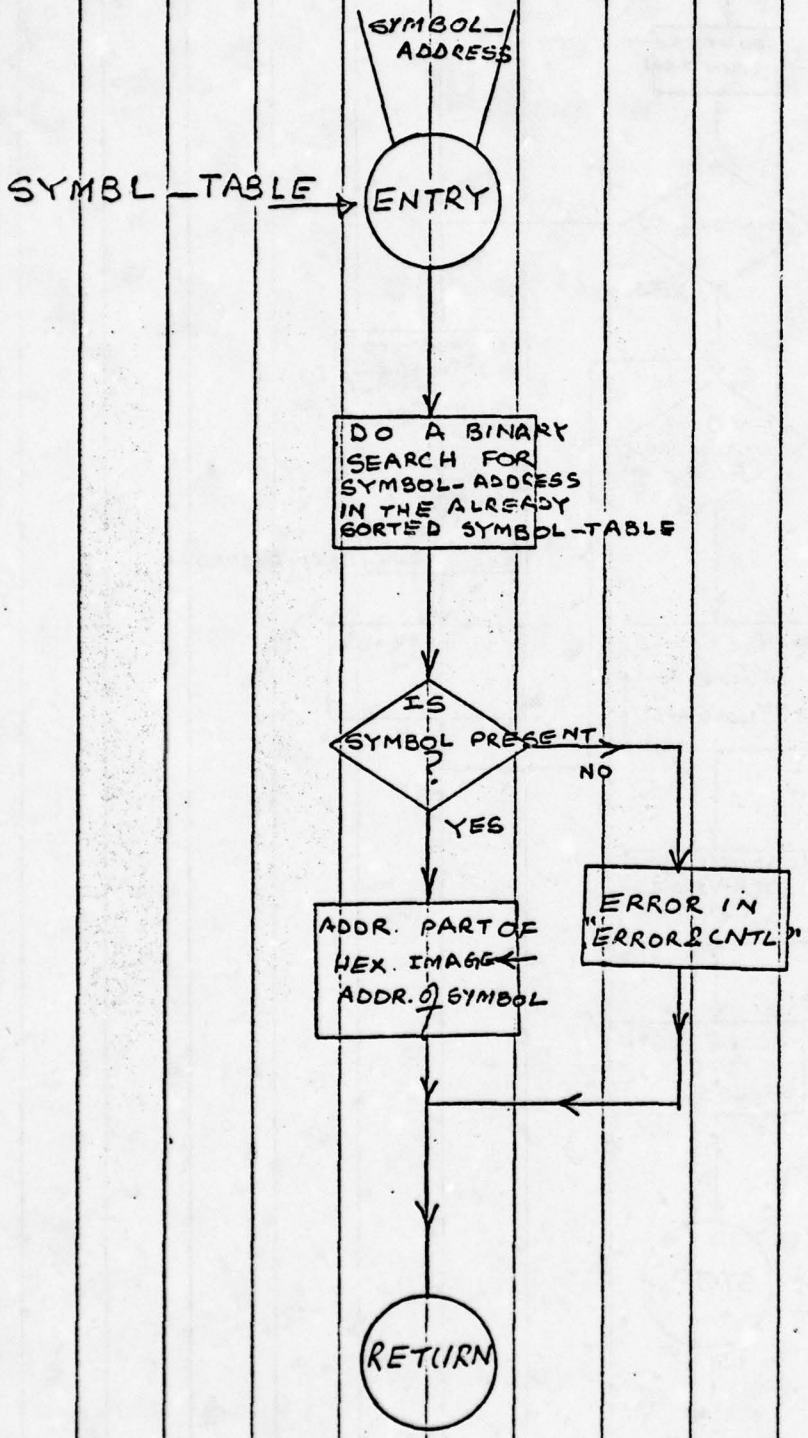
6) EVALUATE - evaluate operands
   Output - operand value

"Average quality" design from Group A; Pass 2, Flowchart condition

A3
C.1




WRITE HEX-IMAGE ON OUTPUT FILE
COUNTER ← COUNTER+1
COUNTER ≤ 38
NO
YES
LEADING RECORD 2ND WORD ← COUNTER TRAIL'N 1ST WORD ← 0
TRAILING 2ND ← 0 3RD ← ADDR OF NEXT INSTR BEGINNING IN NEXT RECORD
COUNTER ← 0
END OF (OUTPUT OR BASS1) FILE
NO
YES
.8
TRAILOR RECORD 1ST WORD ← 0 2ND WORD ← 0
LEADIND REC-2ND WORD ← COUNTER
CLOSE BOTH THE FILES.
STOP


SYMBOL_ADDRESS
SYMBL_TABLE
ENTRY
DO A BINARY SEARCH FOR SYMBOL-ADDRESS IN THE ALREADY SORTED SYMBOL-TABLE
IS SYMBOL PRESENT ?
NO
YES
ERROR IN "ERROR&CNTL"
ADDR. PART OF HEX. IMAGE OF ADDR. OF SYMBOL
RETURN

```
PASS_TWO: PROCEDURE;
   DO WHILE RUN_CODE = CONTINUE EXECUTION CODE;
      READ RECORD IF END OF FILE THEN RUN CODE ← HALT TERMINATION CODE;
      ELSE DO;
         DETERMINE INSTRUCTION_CLASS;
         DO CASE (INSTRUCTION_CLASS);
            CASE 0:
               CALL PROCESS_SYMBOLIC_MACHINE_INSTRUCTION;
            CASE 1:
               CALL PROCESS_CONSTANST_DEFINITIONS;
            CASE 2:
               CALL PROCESS_ASSEMBLER_INSTRUCTIONS;
            CASE 3:
               CALL PROCESS_INVALID_INSTRUCTIONS;
         END CASE;
      END;
      PRINT PASS_TWO STATUS INFORMATION;
      IF OUTPUT THE OUTPUT BUFFER IF NOT EMPTY;
      1ST WORD ← 0
      2ND WORD ← 0
      3RD WORD ← ABS_ADDR OF STARTING INSTR.
   END PASS_TWO;
```

"Average quality" design from Group B, Pass 2, PDL condition

```
PROCESS_ASSEMBLER_INSTRUCTIONS: PROCEDURE;
    CALL OUTPUT;
    RETURN TO CALLER;
END PROCESS_ASSEMBLER_INSTRUCTIONS;
```

```
PROCESS_SYBOLIC_MACHINE-INSTRUCTIONS: PROCEDURE;
 IF ØP=B THEN IF ADDR2=SYMBOLIC THEN CALL SEARCH_SYMBOL_TABLE(HAD,ADDR
      ELSE HAD_←ADDR2;
   ELSE IF ADDR1 = SYMBOLIC THEN CALL SEARCH_SYMBOL_TABLE(HAD_,ADDR1)
      ELSE HAD_←ADDR1;
   INSTR←HOPII HAD_;
   CALL ØUTPUT;
   CALL LOAD (INSTR, HLOC);
      RETURN TO CALLER;
   END PROCESS_SYMBOLIC_MACHINE_INSTRUCTION;
```

```
PROCESS_INVALID_INSTRUCTION: PROCEDURE;
    CALL OUTPUT;
    CALL LOAD (INSTR, HLOC);
    RETURN TO CALLER;
END PROCESS_INVALID_INSTRUCTION;
```

```
PROCESS_CONSTANST_DEFINITION: PROCEDURE
  IF OP = DC THEN DO;
    CALL OUTPUT;
    CALL LOAD (INSTR, HLOC);
  END;
  ELSE DO;
    IF ADDR1 = SYMBOLIC THEN CALL SEARCH_SYMBOL_TABLE (HAD, ADDR1);
    ELSE HAD ← ADDR1;
    IF ADDR2 = SYMBOLIC THEN CALL SEARCH_SYMBOL_TABLE (HAD, ADDR2);
    ELSE HAD ← ADDR2;
    INSTR ← LAST 10 BITS OF ADDR1 || LAST 6 BITS OF ADDR2;
    CALL OUPUT;
    CALL LOAD (INSTR, HLOC;
  END;
  END PROCESS_CONSTANST_DEFINITIONS;
```

```
LOAD: PROCEDURE (INSTR, HLOC);
  IF 2ND WORD OF BUFFER=38 OR HLOC ≠ 2ND WORD + 1ST WORD OF BUFFER OR
     IF LOAD IS CALLED THE 1ST TIME
     THEN DO;
        IF LOAD IS NOT CALLED THE 1ST TIME THEN OUTPUT BUFFER;
        ELSE SAVE ABS_ADDR OF STARTING INSTRUCTION;
        1ST WORD OF BUFFER ← HLOC;
        2ND WORD OF BUFFER ← 1;
        3RD WORD OF BUFFER ← INSTR;
     END;
     ELSE DO;
        INCREMENT 2ND WORD OF BUFFER BY 1;
        (2ND WORD + 2)-TH WORD OF BUFFER ← INSTR;
     END
  RETURN TO CALLER;
END LOAD;
```

```
SEARCH_SYMBOL_TABLE; PROCEDURE (HAD, AD)
   SEARCH SYMBOL TABLE FOR AD AND RETURN HEX_LOCATION (HAD)
   OF AD TO CALLER; IF SYMBLE NOT FOUND HAD ← 0;
END SEARCH_SYMBOL_TABLE;
```

```
OUTPUT: PROCEDURE.
    PRINT HEX IMAGE, ERROR NOTES, INPUT IMAGE.
END OUTPUT;
```

APPENDIX Q - EXAMPLE OF JOB-GENERATED SIMULATOR PROGRAM

```
                                  80/80 LIST

      00000000011111111112222222222333333333344444444445555555555666666666677777777778
      12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
      CPU:  PROCEDURE OPTIONS (MAIN);                                          RWR   20
        /*                                                                     RWR   30
        COMSC 5113 TERM PROJECT --- SIMULATION OF THE DATA GENERAL             RWR   40
        MICRONOVA COMPUTER ON THE IBM SYSTEM 370                               RWR   50
                                                                               RWR   60
        AUTHOR: [redacted]                                                     RWR   70
        DUE DATE: 4-20-77                                                      RWR   80
                                                                               RWR   90
                                                                               RWR  100
        STATEMENT OF THE PROBLEM:  TO IMPLEMENT AND TEST THE MICRONOVA         RWR  110
        COMPUTER BY DATA GENERAL.  THE INSTRUCTION SETS USED WHERE             RWR  120
        TWO ACCUMULATOR, MEMORY REFERENCE, STACK INSTRUCTIONS, AND             RWR  130
        INTERRUPT INSTRUCTIONS.  A MORE DETAILED DESCRIPTION OF THE COMPUTERRWR  140
        ACCOMPANIES THIS PROJECT AND A DETAILED DESCRIPTION OF THE LANGUAGE RWR  150
        CAN BE FOUND IN THE DATA GENERAL MICRONOVA USER MANUAL.                RWR  160
                                                                               RWR  170
        INPUT REQUIREMENTS:  ON EACH DATA CARD                                 RWR  180
                                                                               RWR  190
              COL 1-4 CONTAIN THE STARTING LOCATION TO LOAD CODE               RWR  200
                                                                               RWR  210
              COL 5-8 CONTAIN THE MAXIMUM NUMBER OF WORDS TO BE LOADED         RWR  220
                (A MAX OF 18)                                                  RWR  230
                                                                               RWR  240
              IF A RECORD HAS ZEROS IN FIELDS ONE AND TWO IT IS ASSUMNED TO BE THE
              END OF THE DATA WITH THE STARTING LOCATION IN THE NEXT F(4,0) FIELD.
                                                                               RWR  250
                                                                               RWR  260
      1                                                                        RWR  270
                                                                               RWR  280
                                                                               RWR  290
                                                                               RWR  300
        PROCEDURES USED:                                                       RWR  310
                                                                               RWR  320
              CPU              -A MAIN PROCEDURE THAT CONTROLS THE SIMULATOR.  RWR  330
                               INSTRUCTION FETCH IS DONE HERE.                 RWR  340
                                                                               RWR  350
              CVB              A FUNCTION THAT CONVERTS A CHAR(1) HEX INPUT TO RWR  360
                               A BIT(4) BINARY VALUE                           RWR  370
                                                                               RWR  380
              CVH              A FUNCTION THAT CONVERTS A BIT(4) BINARY        RWR  390
                               VALUE TO A CHAR(1) HEX VALUE                    RWR  400
                                                                               RWR  410
              LOAD             -A PROCEDURE TO INITIALIZE THE SIMULATOR AND    RWR  420
                               LOAD MEMORY                                     RWR  430
                                                                               RWR  440
              TRACE_DUMP       -A PROCEDURE TO EXECUTE A TRACE INSTRUCTION     RWR  450
                                                                               RWR  460
              DUMP             PRINTS A MEMORY DUMP                            RWR  470
                                                                               RWR  480
              TRACE            PRINTS A TRACE                                  RWR  490
                                                                               RWR  500
              EXC              A PROCEDURE TO DECODE AND EXECURE INSTRUCTIONS  RWR  510
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
                                                                                RWR  520
          EFFECTIVE        A PROCEDURE TO COMPUTE AN EFFECTIVE ADDRESS          RWR  530
                                                                                RWR  540
          SHIFT            A PROCEDURE TO PERFORM THE SPECIFIED SHIFT           RWR  550
                           OPERATION                                            RWR  560
                                                                                RWR  570
          CARRY            A PROCEDURE TO INITIALIZE THE CARRY BIT              RWR  580
                                                                                RWR  590
          SKIP             A PROCEDURE TO PERFORM THE SPECIFIED SKIP OPERATION
                                                                                RWR  610
          IND              A PROCEDURE TO COMPUTE THE INDIRECT ADDRESS          RWR  620
                                                                                RWR  630
                                                                                RWR  640
    */                                                                          RWR  650
1    /*                                                                         RWR  660
    MICRO NOVA SIMULATOR VARIABLES                                              RWR  670
                                                                                RWR  680
                                                                                RWR  690
      SYMBOL          DIMENSION DESCRIPTION                                     RWR  700
      AA              4,16      ACCUMULATORS                                    RWR  710
      BB              3,64,16   DEVICE BUFFERS                                  RWR  720
      MM              32768,16  MEMORY                                          RWR  730
      A               16        ACTIVE TRACE LOCATION                           RWR  740
      B               16        SIMULATOR CONTROL                               RWR  750
                                1  CYCLE COUNT EXIT                             RWR  760
                                2  DUMP THEN CONTINUE                           RWR  770
                                3  DUMP THEN EXIT                               RWR  780
                                4  REAL TIME CLOCK CURRENT VALUE                RWR  790
                                5  TRACE ON INDICATOR                           RWR  800
      C               1         CARRY BIT                                       RWR  810
      D               8         INTERRUPT FLAGS                                 RWR  820
                                1  INTERRUPT ENABLE/DISABLE                     RWR  830
                                2  INTERRUPT ENABLE STATE CHANGE                RWR  840
                                3  INTERRUPT PENDING                            RWR  850
                                4  STACK INTERRUPT                              RWR  860
                                5  REAL TIME CLOCK INTERRUPT                    RWR  870
                                6  I/O INTERRUPT                                RWR  880
                                7  SKIP INDICATOR                               RWR  890
                                8  JMP ON JSR INSTR.                            RWR  900
                                9  PRIMARY DELAY FLAB ON MSKO                   RWR  910
                               10 SECONDARY DELAY FLAG ON MSKO                  RWR  920
      E               SCALAR    EFFECTIVE ADDRESS                               RWR  930
      F               15        FRAME POINTER                                   RWR  940
      G               15        STACK POINTER                                   RWR  950
      H                         STATUS                                          RWR  960
                                1  NOT USED                                     RWR  970
                                2  MEMORY OUT OF RANGE                          RWR  980
                                3  DIVIDE OVERFLOW                              RWR  990
      I               16        INSTRUCTION REGISTER                            RWR 1000
      J               SCALAR    INDES FOR CYCLE TABLE TCYCLE                    RWR 1010
      K               16        INTERMEDIATE REGISTER                           RWR 1020
      L               16        RIGHT EXTENSION FOR K                           RWR 1030
      M               16        PRIORITY MASK                                   RWR 1040
      N               16        PRIORITY MASK                                   RWR 1050
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
      P                 15        LOCATION COUNTER                                RWR 1060
      Q                 16        INTERMEDIATE ADDRESS                            RWR 1070
      R                 64        INTERRUPT REQUEST BY DEVICE                     RWR 1080
      S                 17        SHIFTER                                         RWR 1090
      T                           INTERMEDIATE RESULTS OR TEMPORARY STORAGE       RWR 1100
      J                 64        BUSY FLAGS                                      RWR 1110
      V                 64        DONE FLAGS                                      RWR 1120
      W                 64        DEVICE CODES ON                                 RWR 1130
      X                                                                           RWR 1140
      Y                 64        DEVICE PRESENT MASK                             RWR 1150
      CLK               2         RESET VALUE, "RANDOM RESET VALUE"               RWR 1160
      CT                SCALAR    INDIRECT ACCESS COUNT                           RWR 1170
      CYC               4         CYCLE LINIT, CURRENT CYCLE COUNT, CYCLE         RWR 1180
                                  COUNT FOR CURRENT INSTRUCTION, REAL TIME        RWR 1190
                                  CLOCK CURRENT VALUE                             RWR 1200
      DEL               2         PRIMARY, SECONDARY DELAYS                       RWR 1210
      KEY               5         CONSOLE (HAND HELD) FUNCTION ID                 RWR 1220
      LOOP              SCALAR    COUNTER FOR SAV, RET INSTRUCTIONS               RWR 1230
      MSIZ              SCALAR    MEMORY SIZE (MUST BE A POWER OF 2)              RWR 1240
      PT                SCALAR    DECIMAL POINT INTENSIFY                         RWR 1250
      RUN               SCALAR    MACHINE RUN STATE                               RWR 1260
      TCYC_E            10        INSTRUCTIONS                                    RWR 1270
      */                                                                          RWR 1280
    1                                                                             RWR 1290
      DCL                                                                         RWR 1300
      TCOUNT FIXED BINARY INITIAL(20),                                            RWR 1310
      P BIT (15) INITIAL ('0'B),                                                  RWR 1320
      I BIT (16) INITIAL ('0'B),                                                  RWR 1330
      G BIT (15) INITIAL ('0'B),                                                  RWR 1340
      F BIT (15) INITIAL ('0'B),                                                  RWR 1350
      A BIT(15) INITIAL ('0'B),                                                   RWR 1360
      B BIT(16) INITIAL ('0'B),                                                   RWR 1370
      C BIT (1) INITIAL ('0'B),                                                   RWR 1380
      AA (0:3) BIT(16),                                                           RWR 1390
      D BIT(11) INITIAL ('0'B),                                                   RWR 1400
      (M,N) BIT(16) INITIAL ('0'B),                                               RWR 1410
      H BIT(4) INITIAL ('0'B),                                                    RWR 1420
      CVH ENTRY (BIT(*)) RETURNS (CHAR(1)),                                       RWR 1440
      CVB ENTRY (CHAR(*)) RETURNS (BIT(4)),                                       RWR 1450
      STCYCL (0:9) FIXED BINARY INITIAL (5,12,15,17,19,24,36,39,207,295),         RWR 1460
      SCYC (0:3) FIXED BINARY INITIAL (10000,0,0,1200),                           RWR 1470
      T FIXED BINARY,                                                             RWR 1490
      MM (0:512) CHAR(4) INITIAL((511)('0000')),
      MSIZ FIXED BINARY INITIAL (511),
      K BIT(16),                                                                  RWR 1500
       LOOP FIXED BINARY,                                                         RWR 1520
       BB (0:3,0:63) BIT(16),                                                     RWR 1530
       L BIT(16),                                                                 RWR 1540
       Q  BIT(16),                                                                RWR 1550
       R BIT (64),                                                                RWR 1560
       S  BIT(17),                                                                RWR 1570
       (U,V,W,X,Y) BIT(64),                                                       RWR 1580
      E FIXED BINARY                                                              RWR 1590
      ;                                                                           RWR 1600
```

```
                                                
```

```
      0000000001111111111222222222233333333344444444445555555555666666666677777777778
      1234567890123456789012345678901234567890123456789012345678901234567890123456789
CARD
 217              SUBSTR(D,9,2)='00'B;                                          RWR 2140
 218    0   /* REINITIALIZE THE PRIORITY MASK */                                RWR 2150
 219              M=N;                                                          RWR 2160
 220            END;                                                            RWR 2170
 221    0   /* TEST THE SKIP ON INDICATOR */                                    RWR 2180
 222          IF SUBSTR(D,7,1)='1'B THEN                                        RWR 2190
 223              DO;                                                           RWR 2200
 224                UNSPEC(T)='0'B||P;                                          RWR 2210
 225                T=MOD((1+T),32768);                                         RWR 2220
 226                K=UNSPEC(T);                                                RWR 2230
 227                SUBSTR(D,7,1)='0'B;                                         RWR 2240
 228    0   /* CHECK FOR MEMORY OVERFLOW */                                     RWR 2250
 229              IF MSIZ > T THEN P=SUBSTR(K,2,15);                            RWR 2260
 230              ELSE IF MSIZ=32768 THEN                                       RWR 2270
 231                  DO;                                                       RWR 2280
 232                    P='0'B;                                                 RWR 2290
 233                    PUT LIST ('ERROR 3');                                   RWR 2300
 234                    CALL DUMP;                                              RWR 2310
 235                    END;                                                    RWR 2320
 236                ELSE DO;                                                    RWR 2330
 237                    P='1111111111111111'B;                                  RWR 2340
 238                    PUT LIST ('ERROR 4');                                   RWR 2350
 239                    CALL DUMP;                                              RWR 2360
 240                    END;                                                    RWR 2370
 241              END;                                                          RWR 2380
 242          IF SUBSTR(D,2,1)='1'B  THEN SUBSTR(D,2,1)='0'B;                   RWR 2390
 243    0   /* CHECK FOR A REAL TIME CLOCK INTERRUPT */                         RWR 2400
 244          IF SUBSTR(D,11,1) THEN DO;                                        RWR 2410
 245            SCYC(3)=SCYC(3)-SCYC(2);                                        RWR 2420
 246            IF SCYC(3)<=0 THEN DO;                                          RWR 2430
 247                SUBSTR(D,5,1)='1'B;                                         RWR 2440
 248                SCYC(3)=1200;                                               RWR 2450
 249                END;                                                        RWR 2460
 250            END;                                                            RWR 2470
 251                                                                            RWR 2480
 252    0   /* CHECK FOR AN INTERRUPT ENABLE */                                 RWR 2490
 253          ELSE IF SUBSTR(D,1,1)&(SUBSTR(D,4,1)|SUBSTR(D,5,1)                RWR 2500
 254                                 |SUBSTR(D,6,1))                            RWR 2510
 255              THEN DO;                                                      RWR 2520
 256                SUBSTR(D,1,1)='0'B;                                         RWR 2530
 257                SUBSTR(D,3,1)='0'B;                                         RWR 2540
 258                  UNSPEC(J)='0'B||P;                                        RWR 2550
 259                  J=J+1;                                                    RWR 2560
 260                  P=SUBSTR((UNSPEC(J)),2,15);                               RWR 2570
 261                CALL MAC ('0'B,'S','0'B||P);                                RWR 2580
 262                E=5-INDEX (SUBSTR(D,4,3),'1'B);                             RWR 2590
 263    0           CALL INC(E);                                                RWR 2600
 264    0   /* CHECK FOR MEMORY OVERFLOW */                                     RWR 2610
 265              K=UNSPEC(E);                                                  RWR 2620
 266          IF MSIZ > E THEN P=SUBSTR(K,2,15);                                RWR 2630
 267              ELSE IF MSIZ=32768 THEN                                       RWR 2640
 268                  DO;                                                       RWR 2650
 269                    P='0'B;                                                 RWR 2660
 270                    PUT LIST ('ERROR 5');                                   RWR 2670
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
                    CALL DUMP;                                                  RWR 2680
                    END;                                                        RWR 2690
                 ELSE DO;                                                       RWR 2700
                    P='1111111111111111'B;                                      RWR 2710
                    PUT LIST ('ERROR 5');                                       RWR 2720
                    CALL DUMP;                                                  RWR 2730
                    END;                                                        RWR 2740
          SUBSTR(D,8,1)='1'B;                                                   RWR 2750
          END;                                                                  RWR 2760
      END;                                                                      RWR 2770
    1 TRACE_DUMP:PROCEDURE;                                                     RWR 2780
    0 /* THIS PROCEDURE EXECUTES A TRACE/DUMP INSTRUCTION */                    RWR 2790
         DCL AC FIXED BINARY;                                                   RWR 2800
        DCL COUNT FIXED BINARY;                                                 RWR 2810
          UNSPEC(AC)='00000000000000'||SUBSTR(I,4,2);                           RWR 2820
          UNSPEC(COUNT)='00000000000'||SUBSTR(I,6,5);                           RWR 2830
    0 /* IF TRACE REQUEST IS NOT ON THE CHECK THE THE COUNTER, IF IT IS         RWR 2840
      / THEN REUURN, ELSE ASSIGN LOCATION COUNTER TO ACTIVE TRACE LOCATION,RWR 2850
      TURN ON THE TRACE REQUEST, INITIALIZE THE TRACE COUNT, AND RETURN */ RWR 2860
      IF AC=0 THEN                                                              RWR 2870
          IF SUBSTR(B,5,1)='0'B THEN                                            RWR 2880
              IF COUNT=0 THEN RETURN;                                           RWR 2890
              ELSE DO;                                                          RWR 2900
                  A=P;                                                          RWR 2910
                  SUBSTR(B,5,1)=('1'B);                                         RWR 2920
                  TCOUNT=COUNT;                                                 RWR 2930
                  RETURN;                                                       RWR 2940
                  END;                                                          RWR 2950
          ELSE IF A¬=P THEN RETURN;                                             RWR 2960
              ELSE DO;                                                          RWR 2970
                  COUNT=COUNT-1;                                                RWR 2980
                     SUBSTR(I,6,5)=UNSPEC(MOD(COUNT,31));                       RWR 2990
                  CALL MAC ('0'B||P,'S',I);                                     RWR 3000
                  IF COUNT=0 THEN                                               RWR 3010
                      DO;                                                       RWR 3020
                          A=('0'B);                                             RWR 3030
                          SUBSTR(B,5,1)='0'B;                                   RWR 3040
                          END;                                                  RWR 3050
                  RETURN ;                                                      RWR 3060
                  END;                                                          RWR 3070
    0 /* IF THE ACCUMULATOR=1 THEN SET THE ACTIVE TRACE LOCATION TO 0 AND      RWR 3080
      TURN OFF THE TRACE REQUEST */                                             RWR 3090
      ELSE IF AC=1 THEN                                                         RWR 3100
          DO;                                                                   RWR 3110
              A='0'B;                                                           RWR 3120
              SUBSTR(B,5,1)='0'B;                                               RWR 3130
              RETURN;                                                           RWR 3140
          END;                                                                  RWR 3150
    0 /* IF THE ACCUMULATOR=2 THEN PERFORM A DUMP */                            RWR 3160
      ELSE IF AC=2 THEN                                                         RWR 3170
          DO;                                                                   RWR 3180
              CALL DUMP;                                                        RWR 3190
              RETURN;                                                           RWR 3200
              END;                                                              RWR 3210
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
 0   /* IF ACCUMULATOR=3 THEN DUMP AND SET THE RUN CODE TO 0 */                RWR 3220
     ELSE IF AC=3 THEN                                                         RWR 3230
         DO;                                                                   RWR 3240
             CALL DUMP;                                                        RWR 3250
             RUN=0;                                                            RWR 3260
             RETURN;                                                           RWR 3270
     RETURN;                                                                   RWR 3280
     END TRACE_DUMP;                                                           RWR 3290
 1                                                                             RWR 3300
     TRACE: PROCEDURE;                                                         RWR 3310
 0   /* TRACE PROCEDURE LISTS THE VALUES OF SEVERAL VARIABLES VERY             RWR 3320
     VARIABLES IMPORTANT TO THE SIMULATOR FOR ERROR CHECKING */                RWR 3330
 0   /* TRACE MAY BE CALLED FORM TRACE_DUMP AND DUMP PROCEDURES */             RWR 3340
      DCL TC CHAR(1);                                                          RWR 3350
     DCL (TP,TI,TF,TG,TM) CHAR(4),TD CHAR(3),TH CHAR(1),TAC(4) CHAR(4);        RWR 3360
                                                                               RWR 3370
     TP = CVH('0'B||SUBSTR(P,1,3))||CVH(SUBSTR(P,4,4))||CVH(SUBSTR(P,8,4))RWR 3380
             ||CVH(SUBSTR(P,12,4));                                            RWR 3390
                                                                               RWR 3400
     TI = CVH(SUBSTR(I,1,4))||CVH(SUBSTR(I,5,4))||CVH(SUBSTR(I,9,4))           RWR 3410
             ||CVH(SUBSTR(I,13,4));                                            RWR 3420
                                                                               RWR 3430
     TF = CVH('0'B||SUBSTR(F,1,3))||CVH(SUBSTR(F,4,4))||                       RWR 3440
           CVH(SUBSTR(F,8,4))||CVH(SUBSTR(F,12,4));                            RWR 3450
     TG = CVH('0'B||SUBSTR(G,1,3))||CVH(SUBSTR(G,4,4))||                       RWR 3460
           CVH(SUBSTR(G,8,4))||CVH(SUBSTR(G,12,4));                            RWR 3470
                                                                               RWR 3480
     TM = CVH(SUBSTR(M,1,4))||CVH(SUBSTR(M,5,4))||CVH(SUBSTR(M,9,4))||         RWR 3490
             CVH(SUBSTR(M,13,4));                                              RWR 3500
                                                                               RWR 3510
     TD = CVH('000'B||SUBSTR(D,1,1))||CVH(SUBSTR(D,2,4))||                     RWR 3520
             CVH(SUBSTR(D,6,4));                                               RWR 3530
                                                                               RWR 3540
     TH = CVH(SUBSTR(H,1,4));                                                  RWR 3550
                                                                               RWR 3560
     TC = CVH('000'B||C);                                                      RWR 3570
                                                                               RWR 3580
     DO T=1 TO 4;                                                              RWR 3590
        J=T-1;                                                                 RWR 3600
         TAC(T)=CVH(SUBSTR(AA(J),1,4))||CVH(SUBSTR(AA(J),5,4))||               RWR 3610
                 CVH(SUBSTR(AA(J),9,4))||CVH(SUBSTR(AA(J),13,4));              RWR 3620
         END;                                                                  RWR 3630
                                                                               RWR 3640
        PUT SKIP (2);                                                          RWR 3650
     PUT SKIP EDIT ('CYC CT:',SCYC(1),'LOC:',TP,'IR:',TI,'FP:',TF,             RWR 3660
             'SP:',TG,'C:',TC,'AC0:',TAC(1),'AC1:',TAC(2),'AC2:',TAC(3),       RWR 3670
             'AC3:',TAC(4),'FLG:',TD,'M:',TM,'ST:',TH)                         RWR 3680
                 (A, F(7,0),X(1),(26)(A,X(1)));                                RWR 3690
                                                                               RWR 3700
     END TRACE;                                                                RWR 3710
 1                                                                             RWR 3720
     DUMP: PROCEDURE;                                                          RWR 3730
 0   /* DUMP IS A PROCEDURE CALLED FROM TRACE_DUMP TO PRINT THE CONTENTS       RWR 3740
     OF MEMORY */                                                              RWR 3750
```

```
                                                
```

```
      0000000001111111111222222222233333333344444444445555555555666666666677777777778
      1234567890123456789012345678901234567890123456789012345678901234567890123456789
CARD
  433           MM(ADR)=CVH(SUBSTR(R,1,4))||CVH(SUBSTR(R,5,4))||                    RWR 4300
  434               CVH(SUBSTR(R,9,4))||CVH(SUBSTR(R,13,4));                        RWR 4310
  435         K=R;                                                                  RWR 4320
  436       RETURN;                                                                 RWR 4330
  437       END MAC;                                                                RWR 4340
  438  1                                                                            RWR 4350
  439       CVB:PROCEDURE (R) RETURNS (BIT(4));                                     RWR 4360
  440  0    /* CVB IS A PROCEDURE THAT CONVERTS A HEX NUMBER IN CHAR(1) FORMAT     RWR 4370
  441       TO A BIT(4) NUMBER AND RETURNS IT TO THE PLACE OF INVOCATION */         RWR 4380
  442       DCL R CHAR(*),B BIT(4),                                                 RWR 4390
  443             HEX CHAR(16) INITIAL ('0123456789ABCDEF');                        RWR 4400
  444       DCL K BIT(16),J FIXED BINARY;                                           RWR 4410
  445         J=INDEX(HEX,R)-1;                                                     RWR 4420
  446        K=UNSPEC(J);                                                           RWR 4430
  447         B=SUBSTR(K,13,4);                                                     RWR 4440
  448       RETURN (B);                                                             RWR 4450
  449       END CVB;                                                                RWR 4460
  450  1                                                                            RWR 4470
  451       CVH: PROCEDURE (B) RETURNS(CHAR(1));                                    RWR 4480
  452       DCL B BIT(*),H CHAR(1),HEX CHAR(16) INITIAL ('0123456789ABCDEF');       RWR 4490
  453        DCL R FIXED BINARY;                                                    RWR 4500
  454      /* CVH CONVERTS A BIT(4) REPRESENTATION TO A CHAR(1) */                  RWR 4510
  455       UNSPEC(R)='000000000000'B||B;                                           RWR 4520
  456       H=SUBSTR(HEX,R+1,1);                                                    RWR 4530
  457        RETURN (H);                                                           RWR 4540
  458       END CVH;                                                                RWR 4550
  459  1                                                                            RWR 4560
  460       LOAD: PROCEDURE;                                                        RWR 4570
  461       DCL (START,NUMBER,JJ) FIXED BINARY;                                     RWR 4580
  462         DCL (A1,A2) CHAR(4);                                                  RWR 4590
  463      /*  LOAD SUBROUTINE SETS THE INITIAL CONDITIONS OF THE SIMULATOR */     RWR 4600
  464       SUBSTR(D,8,1)='1'B;                                                     RWR 4610
  465         U='0'B;                                                             RWR 4620
  466         V='0'B;                                                             RWR 4630
  467        DO J=0 TO 3;                                                         RWR 4640
  468          AA(J)='0'B;                                                        RWR 4650
  469         END;                                                                RWR 4660
  470        DO WHILE ('1'B);                                                     RWR 4670
  471          GET EDIT (A1,A2)(A(4),A(4));                                       RWR 4680
  472         UNSPEC(START)=CVB(SUBSTR(A1,1,1))||CVB(SUBSTR(A1,2,1))||            RWR 4690
  473           CVB(SUBSTR(A1,3,1))||CVB(SUBSTR(A1,4,1));                         RWR 4700
  474                                                                             RWR 4710
  475           UNSPEC(NUMBER)=CVB(SUBSTR(A2,1,1))||CVB(SUBSTR(A2,2,1))||         RWR 4720
  476           CVB(SUBSTR(A2,3,1))||CVB(SUBSTR(A2,4,1));                         RWR 4730
  477  0    /* CHECK FOR MEMORY "OVER LOAD" */                                     RWR 4740
  478           IF START+NUMBER > MSIZ THEN DO;                                   RWR 4750
  479                 PUT SKIP LIST ('ATTEMPT TO LOAD PAST MEMORY SIZE');         RWR 4760
  480                 RUN=0;                                                      RWR 4770
  481                 RETURN;                                                     RWR 4780
  482                 END;                                                        RWR 4790
  483        /*  CHECK FOR AN END OF PROGRAM */                                   RWR 4800
  484          IF (START=0) & (NUMBER=0) THEN DO;                                 RWR 4810
  485           GET EDIT (A1)(A(4));                                              RWR 4820
  486           UNSPEC(START)=CVB(SUBSTR(A1,1,1))||CVB(SUBSTR(A1,2,1))||          RWR 4830
```

```
0000000001111111111222222222233333333334444444444555555555566666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890123456789
CARD
487              CVB(SUBSTR(A1,3,1))||CVB(SUBSTR(A1,4,1));                    RWR 4840
488                P=SUBSTR((UNSPEC(START)),2,15);                            RWR 4850
489                RETURN;                                                    RWR 4860
490                END;                                                       RWR 4870
491            DO JJ= START TO (NUMBER + START - 1);                          RWR 4880
492              GET EDIT (MM(JJ))( A(4));                                    RWR 4890
493              END;                                                         RWR 4900
494            IF NUMBER ¬= 16 THEN GET SKIP;                                 RWR 4910
495            END;                                                           RWR 4920
496     BOTTOM: END LOAD;                                                     RWR 4930
497   1 EXC:PROCEDURE;                                                        RWR 4940
498       DCL DCODE FIXED BINARY;                                             RWR 4950
499       DCL AC FIXED BINARY;                                                RWR 4960
500       DCL (ACD,ACS) FIXED BINARY;                                         RWR 4970
501       DCL (KK,KK1,KK2) FIXED BINARY (31,0),J1 FIXED BINARY INITIAL(0),    RWR 4980
502           K1 BIT(16);                                                     RWR 4990
503       /* ACD IS THE DESTINATION ACCUMULATOR */                            RWR 5000
504       /* ACS IS THE AOURCE ACCUMULATOR */                                 RWR 5010
505                                                                          RWR 5020
506       /* THIS PROCEDURE PERFORMS INSTRUCTION EXECUTION */                 RWR 5030
507       PUT SKIP (2);                                                       RWR 5040
508      PUT SKIP DATA (P);                                                   RWR 5050
509        PUT SKIP DATA (I);                                                RWR 5060
510       J=0;                                                               RWR 5070
511       UNSPEC(J)='0'B||P;                                                 RWR 5080
512       IF MM(J)='0000' THEN DO;                                           RWR 5090
513          RUN=0;                                                          RWR 5100
514          RETURN;                                                         RWR 5110
515          END;                                                            RWR 5120
516        UNSPEC(J)='000C000000'||SUBSTR(I,11,6);                           RWR 5130
517       IF J=62 THEN DO;                                                   RWR 5140
518        CALL TRACE_DUMP;                                                  RWR 5150
519        RETURN;                                                           RWR 5160
520        END;                                                              RWR 5170
521          UNSPEC(ACD)='C0000000000000'B||SUBSTR(I,4,2);                   RWR 5180
522          UNSPEC(ACS)='000000000000000'B||SUBSTR(I,2,2);                  RWR 5190
523                                                                          RWR 5200
524       /* FIRST CHECK FOR A TWO ACCUMULATOR INSTRUCTION */                 RWR 5210
525       IF SUBSTR(I,1,1) THEN DO;                                          RWR 5220
526       /* TWO ACCUMULATOR INSTRUCTIONS */                                 RWR 5230
527          K='0000000000'B||SUBSTR(I,11,6);                                RWR 5240
528          UNSPEC(J)=K;                                                    RWR 5250
529          IF J=8 THEN DO;                                                 RWR 5260
530       /* TRAP */                                                         RWR 5270
531             CALL MAC('C0C000000101010'B,'S','0'B||P);                    RWR 5280
532             E=39;                                                        RWR 5290
533             CALL IND(E);                                                 RWR 5300
534             IF PUN=0 THEN RETURN;                                        RWR 5310
535             P=SUBSTR((UNSPEC(E)),2,15);                                  RWR 5320
536             SUBSTR(D,8,1)='1'B;                                          RWR 5330
537             SCYC(2)=STCYCL(4);                                           RWR 5340
538             RETURN;                                                      RWR 5350
539             END;                                                         RWR 5360
540          SCYC(2)=STCYCL(1);                                              RWR 5370
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
      CALL CARRY;                                                          RWR 5380
   /* CHECK BITS 10 AND 11 OF I TC DECODE THE INSTRUCTUION */               RWR 5390
      UNSPEC(JJ)='0CC0000000000'B||SUBSTR(I,6,3);                          RWR 5400
      IF JJ=1 THEN DC;                                                     RWR 5410
   /* NEGATE */                                                            RWR 5420
         K=¬AA(ACS);                                                       RWR 5430
         UNSPEC(J)=K;                                                      RWR 5440
         KK=J;                                                             RWR 5450
         S=SUBSTR((UNSPEC(KK+1)),16,17);                                   RWR 5460
         IF SUBSTR(S,1,1) THEN SUBSTR(S,1,1)=¬C;                           RWR 5470
            ELSE SUESTR(S,1,1)=C;                                          RWR 5480
         END;                                                              RWR 5490
      ELSE IF JJ=4 THEN DO;                                                RWR 5500
   /* ADD COMPLEMENT */                                                    RWR 5510
         UNSPEC(J)=AA(ACD);                                                RWR 5520
         K=¬AA(ACS);                                                       RWR 5530
         UNSPEC(J1)=K;                                                     RWR 5540
         KK1=J;                                                            RWR 5550
         KK2=J1;                                                           RWR 5560
         KK=KK1+KK2;                                                       RWR 5570
         S=SUBSTR((UNSPEC(KK)),16,17);                                     RWR 5580
         IF SUBSTR(S,1,1) THEN SUBSTR(S,1,1)=¬C;                           RWR 5590
            ELSE SUESTR(S,1,1)=C;                                          RWR 5600
         END;                                                              RWR 5610
      ELSE IF JJ=5 THEN DO;                                                RWR 5620
   /* SUBTRACT */                                                          RWR 5630
         UNSPEC(J1)=¬AA(ACS);                                              RWR 5640
         UNSPEC(J2)=AA(ACD);                                               RWR 5650
         S=SUBSTR((UNSPEC(J1+1+J2)),16,17);                                RWR 5660
         IF SUBSTR(S,1,1) THEN SUBSTR(S,1,1)=¬C;                           RWR 5670
            ELSE SUESTR(S,1,1)=C;                                          RWR 5680
            END;                                                           RWR 5690
      ELSE IF JJ=6 THEN DO;                                                RWR 5700
   /* ADD */                                                               RWR 5710
         UNSPEC(J1)=AA(ACS);                                               RWR 5720
         UNSPEC(J2)=AA(ACD);                                               RWR 5730
         S=SUBSTR((UNSPEC(J1+J2)),16,17);                                  RWR 5740
         IF SUBSTR(S,1,1) THEN SUBSTR(S,1,1)=¬C;                           RWR 5750
            ELSE SUESTR(S,1,1)=C;                                          RWR 5760
            END;                                                           RWR 5770
   /* COMPLEMENT */                                                        RWR 5780
      ELSE IF JJ=0 THEN                                                    RWR 5790
          S='0'B||¬AA(ACS);                                                RWR 5800
   /* MOVE */                                                              RWR 5810
      ELSE IF JJ=2 THEN S=C||AA(ACS);                                      RWR 5820
   /* INCREMENT */                                                         RWR 5830
      ELSE IF JJ=3 THEN DO;                                                RWR 5840
         UNSPEC(J)=AA(ACS);                                                RWR 5850
         J=J+1;                                                            RWR 5860
         S=C||UNSPEC(J);                                                   RWR 5870
         END;                                                              RWR 5880
   /* AND */                                                               RWR 5890
      ELSE IF JJ=7 THEN S='0'B||(AA(ACS)&AA(ACD));                         RWR 5900
      CALL SHIFT;                                                          RWR 5910
```

```
      0000000001111111111222222222233333333334444444444555555555566666666677777777778
      1234567890123456789012345678901234567890123456789012345678901234567890123456789
CARD
 595          C=SUBSTR(S,1,1);                                                    RWR 5920
 596          CALL SKIP;                                                          RWR 5930
 597        ' IF SUBSTR(I,13,1)='0'B THEN AA(ACD)=SUBSTR(S,2,16);                 RWR 5940
 598         RETURN;                                                              RWR 5950
 599          END;                                                                RWR 5960
 600                                                                              RWR 5970
 601                                                                              RWR 5980
 602      /* NOW CHECK FOR MEMORY REFERENCE INSTRUCTIONS */                       RWR 5990
 603          UNSPEC(J)='00CC0000000000'B||SUBSTR(I,2,2);                         RWR 6000
 604          IF J¬=3 THEN CC;                                                    RWR 6010
 605             CALL EFFECTIVE;                                                  RWR 6020
 606             IF SUBSTR(I,6,1) THEN CALL IND(E);                               RWR 6030
 607  0   /* CHECK FOR A REFERENCE OUTSIDE THE BOUNDS OF MEMORY */                RWR 6040
 608        IF E> MSIZ THEN DO;                                                   RWR 6050
 609             PUT SKIP LIST ('ATTEMPT TO REFERENCE BEYOND MEMORY SIZE');       RWR 6060
 610             CALL DUMP;                                                       RWR 6070
 611             RUN=0;                                                           RWR 6080
 612             RETURN;                                                          RWR 6090
 613             END;                                                             RWR 6100
 614                                                                              RWR 6110
 615               IF RUN=0 THEN RETURN;                                          RWR 6120
 616               IF (J=1)|(J=2) THEN DO;                                        RWR 6130
 617                 K=UNSPEC(E);                                                 RWR 6140
 618      /* LOAD OR STORE THE ACCUMULATOR */                                     RWR 6150
 619                 IF SUBSTR(I,2,1) THEN CALL MAC(K,'S',AA(ACD));               RWR 6160
 620                   ELSE CALL MAC(K,'F',AA(ACD));                              RWR 6170
 621                 SCYC(2)=SCYC(2)+STCYCL(2);                                   RWR 6180
 622                 RETURN;                                                      RWR 6190
 623                 END;                                                         RWR 6200
 624             UNSPEC(J)='000000000000000'B||SUBSTR(I,4,2);                     RWR 6210
 625             IF J=0 THEN DO;                                                  RWR 6220
 626      /* JUMP */                                                              RWR 6230
 627                 SCYC(2)=SCYC(2)+STCYCL(2);                                   RWR 6240
 628                 P=SUBSTR((UNSPEC(E)),2,15);                                  RWR 6250
 629                 SUBSTR(C,8,1)='1'B;                                          RWR 6260
 630                 RETURN;                                                      RWR 6270
 631                 END;                                                         RWR 6280
 632             ELSE IF J=1 THEN DO;                                             RWR 6290
 633      /* SUBROUTINE JUMP */                                                   RWR 6300
 634                 UNSPEC(J1)='0'B||P;                                          RWR 6310
 635                 J1=J1+1;                                                     RWR 6320
 636                 K=UNSPEC(J1);                                                RWR 6330
 637                 IF MSIZ <= J1 THEN                                           RWR 6340
 638                    IF MSIZ=32768 THEN AA(3)='0'B;                            RWR 6350
 639                      ELSE AA(3)='1111111111111111'B;                         RWR 6360
 640                   ELSE AA(3)=K;                                              RWR 6370
 641                 SCYC(2)=SCYC(2)+STCYCL(2);                                   RWR 6380
 642                 P=SUBSTR((UNSPEC(E)),2,15);                                  RWR 6390
 643                 SUBSTR(C,8,1)='1'B;                                          RWR 6400
 644                 RETURN;                                                      RWR 6410
 645                 END;                                                         RWR 6420
 646             ELSE DO;                                                         RWR 6430
 647      /* INCREMENT OR DECREMENT AND SKIP */                                   RWR 6440
 648                 K1=UNSPEC(E);                                                RWR 6450
```

```
0000000001111111111222222222233333333334444444444555555555566666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890123456789
CARD
649                  CALL MAC(K1,'F',K);                                    RWR 6460
650                  UNSPEC(J)=K;                                           RWR 6470
651                  IF SUBSTR(I,5,1) THEN J=J-1;                           RWR 6480
652                     ELSE J=J+1;                                         RWR 6490
653                  K=UNSPEC(J);                                           RWR 6500
654                  CALL MAC(K1,'S',K);                                    RWR 6510
655                  IF J=0 THEN DO;                                        RWR 6520
656                    UNSPEC(J)='0'B||P;                                   RWR 6530
657                    J=J+1;                                               RWR 6540
658                    P=SUBSTR((UNSPEC(J)),2,15);                          RWR 6550
659                    SCYC(2)=SCYC(2)+STCYCL(0);                           RWR 6560
660                    END;                                                 RWR 6570
661                  END;                                                   RWR 6580
662               RETURN;                                                   RWR 6590
663            END;                                                        RWR 6600
664                                                                        RWR 6610
665                                                                        RWR 6620
666         ELSE DO;                                                       RWR 6630
667          /* CHECK FOR A STACK INSTRUCTION */                           RWR 6640
668          UNSPEC(J)='000000000'B||SUBSTR(I,11,6);                       RWR 6650
669          IF J=1 THEN DO;                                               RWR 6660
670             UNSPEC(J)='000000000000'B||SUBSTR(I,6,3);                  RWR 6670
671             UNSPEC(AC)='000000000000000'B||SUBSTR(I,4,2);              RWR 6680
672             IF J=0 THEN DO;                                            RWR 6690
673          /* MOVE TO OR FROM FRAME POINTER */                           RWR 6700
674               SCYC(2)=STCYCL(1)+SCYC(2);                               RWR 6710
675               IF SUBSTR(I,9,1) THEN AA(AC)='0'B||F;                    RWR 6720
676                  ELSE F=SUBSTR(AA(AC),2,15);                           RWR 6730
677               RETURN;                                                  RWR 6740
678               END;                                                     RWR 6750
679             IF J=2 THEN DO;                                            RWR 6760
680          /* MOVE TO OR FROM STACK POINTER */                           RWR 6770
681               SCYC(2)=SCYC(2)+STCYCL(1);                               RWR 6780
682               IF SUBSTR(I,9,1) THEN AA(AC)='0'B||G;                    RWR 6790
683                  ELSE G=SUBSTR(AA(AC),2,15);                           RWR 6800
684               RETURN;                                                  RWR 6810
685               END;                                                     RWR 6820
686             IF J=3 THEN DO;                                            RWR 6830
687          /* PUSH OR POP THE ACCUMULATOR */                             RWR 6840
688               SCYC(2)=SCYC(2)+STCYCL(3);                               RWR 6850
689               IF SUBSTR(I,9,1) THEN DO;                                RWR 6860
690                  CALL MAC('0'B||G,'F',AA(AC));                         RWR 6870
691                  UNSPEC(J1)='0'B||G;                                   RWR 6880
692                  J1=J1-1;                                              RWR 6890
693                  G=SUBSTR((UNSPEC(J1)),2,15);                          RWR 6900
694                  RETURN;                                               RWR 6910
695               END;                                                     RWR 6920
696            ELSE DO;                                                    RWR 6930
697               UNSPEC(J1)='0'B||G;                                      RWR 6940
698               J1=J1+1;                                                 RWR 6950
699               G=SUBSTR((UNSPEC(J1)),2,15);                             RWR 6960
700               CALL MAC('0'B||G,'S',AA(AC));                            RWR 6970
701               IF MOD(J1,256) ¬= 0 THEN RETURN;                         RWR 6980
702               SUBSTR(D,3,2)='11'B;                                     RWR 6990
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
                RETURN;                                                         RWR 7000
                END;                                                            RWR 7010
            END;                                                                RWR 7020
        IF J=5 THEN                                                             RWR 7030
            IF SUBSTR(I,9,1) THEN DO;                                           RWR 7040
        /*  RETURN BLOCK */                                                     RWR 7050
                G=F;                                                            RWR 7060
                UNSPEC(J1)='0'B||G;                                             RWR 7070
                J1=J1-1;                                                        RWR 7080
                G=SUBSTR((UNSPEC(J1)),2,15);                                    RWR 7090
                CALL MAC('0'B||G,'F',K);                                        RWR 7100
                C=SUBSTR(K,1,1);                                                RWR 7110
                P=SUBSTR(K,2,15);                                               RWR 7120
                J1=J1-1;                                                        RWR 7130
                G=SUBSTR((UNSPEC(J1)),2,15);                                    RWR 7140
                CALL MAC('0'B||G,'F',AA(3));                                    RWR 7150
                J1=J1-1;                                                        RWR 7160
                G=SUBSTR((UNSPEC(J1)),2,15);                                    RWR 7170
                CALL MAC('0'B||G,'F',AA(2));                                    RWR 7180
                J1=J1-1;                                                        RWR 7190
                G=SUBSTR((UNSPEC(J1)),2,15);                                    RWR 7200
                CALL MAC ('0'B||G,'F',AA(1));                                   RWR 7210
                J1=J1-1;                                                        RWR 7220
                G=SUBSTR((UNSPEC(J1)),2,15);                                    RWR 7230
            CALL MAC('0'B||G,'F',AA(0));                                        RWR 7240
                SUBSTR(D,8,1)='1'B;                                             RWR 7250
                SCYC(2)=SCYC(2)+STCYCL(6);                                      RWR 7260
                RETURN;                                                         RWR 7270
                END;                                                            RWR 7280
            ELSE DO;                                                            RWR 7290
        /* SAVE BLOCK */                                                        RWR 7300
                DO LOOP= 1 TO 5;                                                RWR 7310
                    UNSPEC(J1)='0'B||G;                                         RWR 7320
                    J1=J1+1;                                                    RWR 7330
                    G=SUBSTR((UNSPEC(J1)),2,15);                                RWR 7340
                    IF MOD(J1,256)=0 THEN SUBSTR(D,3,2)='11'B;                  RWR 7350
                    IF LOOP=1 THEN K=AA(0);                                     RWR 7360
                    ELSE IF LOOP=2 THEN K=AA(1);                                RWR 7370
                    ELSE IF LOOP=3 THEN K=AA(2);                                RWR 7380
                    ELSE IF LOOP=4 THEN K='0'B||F;                              RWR 7390
                    ELSE IF LOOP=5 THEN K=C||SUBSTR(AA(3),2,15);                RWR 7400
                    CALL MAC('0'B||G,'S',K);                                    RWR 7410
                    END;                                                        RWR 7420
                F=G;                                                            RWR 7430
                AA(3)='0'B||G;                                                  RWR 7440
                RETURN;                                                         RWR 7450
                END;                                                            RWR 7460
        IF J=6 THEN                                                             RWR 7470
            IF SUBSTR(I,9,1) THEN DO;                                           RWR 7480
                UNSPEC(J)=AA(2);                                                RWR 7490
                KK=J;                                                           RWR 7500
                UNSPEC(J)=AA(1);                                                RWR 7510
                KK1=J;                                                          RWR 7520
                UNSPEC(J)=AA(0);                                                RWR 7530
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
                KK2=J;                                                          RWR 7540
                KK=KK*KK1+KK2*65536;                                            RWR 7550
                AA(0)=SUBSTR((UNSPEC(KK)),1,16);                                RWR 7560
                AA(1)=SUBSTR((UNSPEC(KK)),17,16);                               RWR 7570
                SCYC(2)=SCYC(2)+STCYCL(8);                                      RWR 7580
                RETURN;                                                         RWR 7590
                END;                                                            RWR 7600
             ELSE DO;                                                           RWR 7610
                UNSPEC(J)=AA(0);                                                RWR 7620
                KK1=J;                                                          RWR 7630
                UNSPEC(J)=AA(2);                                                RWR 7640
                KK2=J;                                                          RWR 7650
                IF KK1<KK2 THEN DO;                                             RWR 7660
                   KK=KK1/KK2;                                                  RWR 7670
                   AA(1)=SUBSTR((UNSPEC(KK)),17,16);                            RWR 7680
                   KK=MOD(KK1,KK2);                                             RWR 7690
                   AA(0)=SUBSTR((UNSPEC(KK)),17,16);                            RWR 7700
                   C='0'B;                                                      RWR 7710
                   SCYC(2)=SCYC(2)+STCYCL(9);                                   RWR 7720
                   RETURN;                                                      RWR 7730
                   END;                                                         RWR 7740
                ELSE DO;                                                        RWR 7750
                   SUBSTR(H,3,1)='1'B;                                          RWR 7760
                   C='1'B;                                                      RWR 7770
                   RETURN;                                                      RWR 7780
                   END;                                                         RWR 7790
                END;                                                            RWR 7800
          ELSE PUT SKIP LIST('OP CODE ERROR IN STACK INSTR');                   RWR 7810
       END;                                                                     RWR 7820
   0                                                                            RWR 7830
       /* CHECK FOR AN INTERRUPT INSTRUCTION */                                 RWR 7840
       UNSPEC(J)='0000000000'B||SUBSTR(I,11,6);                                 RWR 7850
       IF J=63 THEN DO;                                                         RWR 7860
          SCYC(2)=SCYC(2)+STCYCL(1);                                            RWR 7870
          UNSPEC(J)='0000000000000'B||SUBSTR(I,6,3);                            RWR 7880
          IF J=0 THEN DO;                                                       RWR 7890
       /* INTERRUPT ENABLE OR DISABLE */                                        RWR 7900
             IF (SUBSTR(D,1,1)='0'B)&(SUBSTR(I,10,1)='1'B) THEN                 RWR 7910
                SUBSTR(D,2,1)='1'B;                                             RWR 7920
             END;                                                               RWR 7930
          ELSE IF J=2 THEN DO;                                                  RWR 7940
       /* RESET BUSY AND DONE FLAGS */                                          RWR 7950
                IF SUBSTR(I,4,2)='00'B THEN DO;                                 RWR 7960
                   V='0'B;                                                      RWR 7970
                   U='0'B;                                                      RWR 7980
                  N='0'B;                                                       RWR 7990
                   END;                                                         RWR 8000
                SUBSTR(D,5,1)=SUBSTR(I,4,1);                                    RWR 8010
                END;                                                            RWR 8020
          ELSE IF J=4 THEN DO;                                                  RWR 8030
       /* REAL TIME CLOCK ENABLE OR DISABLE */                                  RWR 8040
             SUBSTR(D,11,1)='1'B;                                               RWR 8050
             UNSPEC(J)='00000000000000'B||SUBSTR(I,4,2);                        RWR 8060
             N=AA(J);                                                           RWR 8070
```

```
                                                
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
          CALL IND(E);                                                      RWR 8620
            SUBSTR(I,5,1)='0'B;                                             RWR 8630
          DO LOOP=E TO E+16;                                                RWR 8640
             PUT SKIP LIST (MM(LOOP));                                      RWR 8650
             END;                                                           RWR 8660
          END;                                                              RWR 8670
       ELSE PUT SKIP LIST ('INVALID DEVICE CODE FOR DATA OUT');             RWR 8680
       CALL BUSY_DONE;                                                      RWR 8690
             RETURN;                                                        RWR 8700
    1  IOSKIP:  PROCEDURE;                                                  RWR 8710
       JNSPEC(J)='00000000000000'B||SUBSTR(I,9,2);                          RWR 8720
       IF (J=0) & (SUBSTR(U,DCODE,1)='1'B) THEN SUBSTR(D,8,1)='1'B;         RWR 8730
       ELSE IF (J=1) & (SUBSTR(U,DCODE,1)='0'B) THEN SUBSTR(D,8,1)='1'B;    RWR 8740
       ELSE IF (J=2) & (SUBSTR(V,DCODE,1)='1'B) THEN SUBSTR(D,8,1)='1'B;    RWR 8750
       ELSE IF (J=3) & (SUBSTR(V,DCODE,1)='0'B) THEN SUBSTR(D,8,1)='1'B;    RWR 8760
       RETURN;                                                              RWR 8770
       END;                                                                 RWR 8780
    1  BUSY_DONE:  PROCEDURE;                                               RWR 8790
          UNSPEC(J)='00000000000000'B||SUBSTR(I,9,2);                       RWR 8800
       IF J=1 THEN DO;                                                      RWR 8810
       /* START DEVICE */                                                   RWR 8820
          SUBSTR(U,DCODE,1)='1'B;                                           RWR 8830
          SUBSTR(V,DCODE,1)='0'B;                                           RWR 8840
          END;                                                              RWR 8850
       ELSE IF J=2 THEN DO;                                                 RWR 8860
       /* IDLE THE DEVICE */                                                RWR 8870
          SUBSTR(U,DCODE,1)='0'B;                                           RWR 8880
          SUBSTR(V,DCODE,1)='0'B;                                           RWR 8890
          END;                                                              RWR 8900
       ELSE IF J=3 THEN DO;                                                 RWR 8910
       /* PULSE THE IN-OUT BUS CONTROL LINE *****/                          RWR 8920
             END;                                                           RWR 8930
       RETURN;                                                              RWR 8940
       END BUSY_DONE;                                                       RWR 8950
      END;                                                                  RWR 8960
    1  SKIP: PROCEDURE;                                                     RWR 8970
       DCL (J,K,L) FIXED BINARY;                                            RWR 8980
       UNSPEC(J)='0000000000000'B||SUBSTR(I,14,3);                          RWR 8990
          UNSPEC(K)=S;                                                      RWR 9000
          IF J=0 THEN RETURN;                                               RWR 9010
          ELSE IF J=1 THEN SUBSTR(D,7,1)='1'B;                              RWR 9020
          ELSE IF (J=2)&(C='0'B) THEN SUBSTR(D,7,1)='1'B;                   RWR 9030
          ELSE IF (J=3)&(C¬='0'B) THEN SUBSTR(D,7,1)='1'B;                  RWR 9040
          ELSE IF (J=4)&(K=0) THEN SUBSTR(D,7,1)='1'B;                      RWR 9050
          ELSE IF (J=5)&(K¬=0) THEN SUBSTR(D,7,1)='1'B;                     RWR 9060
          ELSE IF (J=6)&((C='0'B)|(K=0)) THEN SUBSTR(D,7,1)='1'B;           RWR 9070
          ELSE IF (J=7)&((C¬='0'B)&(K¬=0)) THEN SUBSTR(D,7,1)='1'B;         RWR 9080
       IF SUBSTR(D,7,1) THEN SCYC(2)=SCYC(2)+STCYCL(0);                     RWR 9090
       RETURN;                                                              RWR 9100
       END SKIP;                                                            RWR 9110
    1  EFFECTIVE: PROCEDURE;                                                RWR 9120
       DCL (J,K,L,N) FIXED BINARY;                                          RWR 9130
       UNSPEC(J)='00000000000000'B||SUBSTR(I,7,2);                          RWR 9140
       IF J=0 THEN UNSPEC(E)='00000000'B||SUBSTR(I,9,8);                    RWR 9150
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
      ELSE IF J=1 THEN DO;                                                  RWR 9160
         UNSPEC(N)='0'B||P;                                                 RWR 9170
         UNSPEC(L)='00000000'B||SUBSTR(I,9,8);                              RWR 9180
         E=N+L;                                                             RWR 9190
         IF SUBSTR(I,9,1) THEN E=E-512;                                     RWR 9200
         END;                                                               RWR 9210
      ELSE IF J=2 THEN DO;                                                  RWR 9220
         UNSPEC(N)=AA(2);                                                   RWR 9230
         UNSPEC(L)='00000000'B||SUBSTR(I,9,8);                              RWR 9240
         E=N+L;                                                             RWR 9250
         IF SUBSTR(I,9,1) THEN E=E-512;                                     RWR 9260
         END;                                                               RWR 9270
      ELSE IF J=3 THEN DO;                                                  RWR 9280
         UNSPEC(N)=AA(3);                                                   RWR 9290
         UNSPEC(L)='00000000'B||SUBSTR(I,9,8);                              RWR 9300
         E=N+L;                                                             RWR 9310
         IF SUBSTR(I,9,1) THEN E=E-512;                                     RWR 9320
         END;                                                               RWR 9330
      E=MOD(E,32768);                                                       RWR 9340
      RETURN;                                                               RWR 9350
      END EFFECTIVE;                                                        RWR 9360
    1 CARRY: PROCEDURE;                                                     RWR 9370
      DCL J FIXED BINARY;                                                   RWR 9380
      UNSPEC(J)='00000000000000'B||SUBSTR(I,11,2);                          RWR 9390
      IF J=1 THEN C='0'B;                                                   RWR 9400
      ELSE IF J=2 THEN C='1'B;                                              RWR 9410
      ELSE IF J=3 THEN C=¬C;                                                RWR 9420
      RETURN;                                                               RWR 9430
      END CARRY;                                                            RWR 9440
    1 SHIFT: PROCEDURE;                                                     RWR 9450
      DCL J FIXED BINARY;                                                   RWR 9460
      UNSPEC(J)='00000000000000'B||SUBSTR(I,9,2);                           RWR 9470
         IF J=0 THEN RETURN;                                                RWR 9480
         SUBSTR(S,1,1)=C;                                                   RWR 9490
      IF J=1 THEN S=SUBSTR(S,2,16)||SUBSTR(S,1,1);                          RWR 9500
      ELSE IF J=2 THEN S=SUBSTR(S,17,1)||SUBSTR(S,1,16);                    RWR 9510
      ELSE IF J=3 THEN S=SUBSTR(S,9,8)||SUBSTR(S,1,8);                      RWR 9520
      RETURN;                                                               RWR 9530
      END SHIFT;                                                            RWR 9540
    END EXC;                                                                RWR 9550
    1 IND:  PROCEDURE (E);                                                  RWR 9560
                                                                            RWR 9570
      DCL E FIXED BINARY;                                                   RWR 9580
      DCL CT FIXED BINARY INITIAL (0), (Q,J) FIXED BINARY,A BIT (16);       RWR 9590
       DCL B BIT(16);                                                       RWR 9600
      /* THIS PROCEDURE COMPUTES AN INDIRECT ADDRESS IF BIT 5 OF THE        RWR 9610
      INSTRUCTION IS A 1 */                                                 RWR 9620
                                                                            RWR 9630
      CT=0;                                                                 RWR 9640
      IF SUBSTR(I,6,1) THEN DO WHILE ('1'B);                                RWR 9650
         A=CVB(SUBSTR(MM(E),1,1))||CVB(SUBSTR(MM(E),2,1))||                 RWR 9660
               CVB(SUBSTR(MM(E),3,1))||CVB(SUBSTR(MM(E),4,1));              RWR 9670
                                                                            RWR 9680
         CT=CT+2;                                                           RWR 9690
```

80/80 LIST

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
CARD
          IF CT>15 THEN DO;                                                    RWR 9700
                RUN=0;                                                         RWR 9710
                CALL TRACE;                                                    RWR 9720
                PUT SKIP LIST ('CHAINING TO LONG IN IND ROUTINE ');            RWR 9730
                RETURN;                                                        RWR 9740
                END;                                                           RWR 9750
          UNSPEC(Q)='0'B||SUBSTR(A,2,15);                                      RWR 9760
          IF (16<=Q) & (31>=Q) THEN DO;                                        RWR 9770
                IF Q<= 23 THEN Q=Q-1;                                          RWR 9780
                ELSE Q=Q+1;                                                    RWR 9790
                B=UNSPEC(Q);                                                   RWR 9800
                CALL MAC((UNSPEC(E)),'S',B);                                   RWR 9810
                CT=CT+1;                                                       RWR 9820
                IF CT > 15 THEN DO;                                            RWR 9830
                   RUN=0;                                                      RWR 9840
                   CALL TRACE;                                                 RWR 9850
                   PUT SKIP LIST ('CHAINING TO LONG IN IND ROUTINE');          RWR 9860
                   RETURN;                                                     RWR 9870
                   END;                                                        RWR 9880
                UNSPEC(E)='0'B||SUBSTR(A,2,15);                                RWR 9890
                IF SUBSTR((UNSPEC(Q)),1,1)='0'B THEN                           RWR 9900
                   RETURN;                                                     RWR 9910
                END;                                                           RWR 9920
          ELSE DO;                                                             RWR 9930
                UNSPEC(E)='0'B||SUBSTR(A,2,15);                                RWR 9940
                IF SUBSTR(A,1,1)='0'B THEN                                     RWR 9950
                   RETURN;                                                     RWR 9960
          END;                                                                 RWR 9970
     END;                                                                      RWR 9980
     END IND;                                                                  RWR 9990
0                                                                              RWR10000
      END CPU;                                                                 RWR10010
                                                                               RWR10020
                                                                               RWR10030

$DATA
```